Interface

2004
8
Contents



[表紙デザイン：㈱プランニング・ロケッツ]





特集

組み込み標準プラットホームをめざして，普及をはかる

新世代TRONアーキテクチャ T-Engine誕生

Cover Story "T-Engine", the next generation TRON architecture is born 35

Interface

2004 8

Contents

連載「はじめて使うμClinux」と「VxWorksを使ったRTOS技術の基礎と応用」は,お休みさせていただきます.

先端要素技術の総合展示会

TECHNO-FRONTIER 2004

北村 俊之

エレクトロニクス，メカトロニクスの要素技術，関連製品を一堂に展示する「TECHNO-FRONTIER 2004」が4月21日(水)～23(金)の3日間，幕張メッセで開催された(写真1)．主催は(社)日本能率協会．本展示会では，「第22回 モータ技術展」，「第19回 電源システム展」，「第17回 EMC・ノイズ対策技術展」，「第13回 モーション・エンジニアリング展」，「第13回 ボード・コンピュータ展」，「第6回 熱対策技術展」，「第5回 高効率・省エネ促進技術展」，「第4回 カーエレクトロニクスデバイス展」，「第4回 パーソナルエリア無線通信技術/Bluetooth Expo」の九つの展示会，および「開発・設計を支援する解析技術」，「リニアアクチュエータと応用技術」，「アドバンストバッテリーシステム」，「静音化設計技術展」，「産学交流技術移転フォーラム」の五つの特別企画，関連企画として「海外部品調達展2004」が同時に開催されるという大規模な展示会となった．全体の総展示規模は561社/1352小間で，最終的な来場者数は3日間で124,781人．今回は，「第13回 ボード・コンピュータ展」および「第4回 パーソナルエリア無線通信技術/Bluetooth Expo」を中心にレポートを行う．

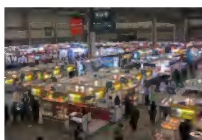


写真1　場内のようす

● 第14回 ボード・コンピュータ展

ボード・コンピュータ展は，組み込み機器，システムを最適に構築するための各種ボードやソフトウェア，開発ツール，周辺機器の実用技術に焦点を絞った専門の展示会である．今回，注目を集めていたのが，「Intel Pentium M」を搭載したボードだった．

アドバネットは，CompactPCI，VME，PCI対応の各種ボード，およびPenitumシリーズ，PowerPC，SHシリーズなどを搭載した各種CPUボードを展示していた．今回展示された製品の中で「A6pci8014」(写真2)は，Pentium Mを搭載したCompactPCI対応のCPUボードで，次世代産業機器に適したCPUパワーを提供しているとのことだった．



写真2　アドバネットのA6pci8014

サンリツオートメーション(写真3)は，各種CompactPCI対応CPUボードを中心とした展示を行っていた．Pentium M搭載の高機能モデル，SC2400シリーズは，来場者の関心も高かったという．同製品は，Pentium M/855GMEとPhoenixBIOSとの組み合わせにより，高い信頼性と安定した製品供給を可能にしているとのことだった．また，PCM拡張をサポートするほか，メザニン・カードにより汎用ボードでは難しいとされていたユーザ・オリエンテッドなカスタムCPUボードの実現も可能だという．

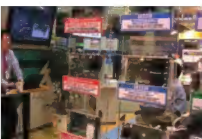


写真3　サンリツオートメーションのブース

アドバンテックは，Pentium M搭載ボード，産業用マザーボード，CompactPCI，XScaleプラットホーム，工業用/産業用パネルPCなど幅広い展示を行っていた．「PCM-9581」(写真4)は，LAN，VGA/LVDS，オーディオ機能を搭載しPentium Mに対応した5.25インチ・シングルボードで，FSB 400MHzのサポート，200/266/333MHzのDDR DIMMのサポートなどを大きな特徴としている．同社では，ボード・レベルからBIOSのカスタマイズ，筐体のカスタマイズなどのシステム・ソリューションまで，幅広い提案を行っているとのことだった．



写真4　アドバンテックのPCM-9581

アバールデータは，CompactPCI，画像入力処理，高速シリアル通信ギガ・チャネルの各製品を展示していた．CompactPCIモジュールは，来場者の関心も高かったとのことである．「ACP-128-1」は，Pentium M/1.6GHzを搭載しており，チップセットには855GMEとICH4を採用している．メモリを最大2Gバイト搭載可能で，RS-232-C，USB，GビットEthernet，GPIOなど多彩なインターフェースを搭載している点が大きな特徴とのこと．さらに，2.5インチHDDと拡張用PMCメザニン・ボードをボード上に搭載可能とのことだった．

ロッキー(写真5)は，同社の主力であるStarFabric製品を中心に，StarGen，BittWare，AccelChip，Gaga各社の製品を展示していた．現在，同社で力を入れているのが，AccelChip社の「AccelChip DSP Synthesis」とのこと．同製品は，MATLABで開発したアルゴリズムから，FPGAで論理合成可能なRTLをダイレクトに自動生成することができる．これによって，FPGA，ASICでDSPアプリケーションを開発する時間が，飛躍的に短縮可能であるという．また，PMCメザニン・ボードのPCIバスを全2重，2.5Gbpsの高速転送でStarFabricに接続可能な，StarGate PMCモジュールも来場者の関心を集めていた．

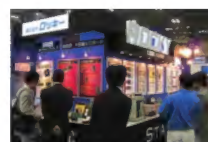


写真5　ロッキーのブース

● 第4回 パーソナルエリア無線通信技術/Bluetooth Expo

Bluetooth Expoは，今回から「パーソナルエリア無線通信技術/Bluetooth Expo」に名称が変更され，Bluetoothなどのパーソナル・エリア無線通信関連の各種デバイス，測定機器，ソフトウェア，認証サービスからソリューションまでを幅広く展示していた．

アジレント・テクノロジーは，BluetoothからUWBに移行する際に最適な，UWBとWLANに関する測定ソリューションを多数展示していた(写真6)．UWBソリューションは，汎用測定器とシミュレータを組み合わせることで，送受信試験へ対応，528MHzのMB-OFDMキャリア信号の生成が可能，6GHzまでのRF，Baseband IQ，差動IQのいずれでも6GHzまでの広域帯信号解析が可能などを特徴としている．また，PSGシリーズの信号発生器は，IEEE802.11などの無線信号を出力できるので，妨害信号出力器としての利用も可能とのことだった．



写真6　アジレント・テクノロジーのWLAN計測機器群

エーアイコーポレーションでは，「ZigBee Ready Starter Bundle」(Helicomm製，写真7)のデモを行っていた．ZigBeeは，ZigBee Allianceによって制定された省電力無線のグローバル規格．同製品を導入することによって，提供された無線モジュールに温度センサおよびI/Oの搭載が可能で，さらに別途IEEE802.15.2 PHY BB/MACのライセンスが取得可能などのメリットがあるとのことだった．



写真7　エーアイコーポレーションのZigBee Ready Starter Bundle

▶ネットワーク＆通信関連の総合展示会

Networld + Interop 2004 Las Vegas

松本 信幸

今年のキーワードは？

5月11日から13日まで，米国Las VegasのLas Vegas convention centerで開催されたNetworld + Interop 2004 Las Vegas(以下N＋I)を視察してきた．

今回のN＋Iでは，新しいソリューションは特になく，従来からあるものを身近にした感が強かった．その中で目を引いたキーワードは，大きく「ワイヤレス」，「VoIP」，「セキュリティ」の三つであった．

また，これまでの展示ではよく見かけたEthernet技術系のブース(Metro Ethernet Forumなど)を始め技術動向的なブースはほとんどなく，製品やサービスの提供を主眼とする出展が多く見うけられた．やや，技術的なものとしてはiLabs程度であったが，そこでは6ベンダ6台のSIP Serverに14ベンダ44台のSIP Phoneが接続され，相互接続を行うというサービス提供を目的としたデモがあった(写真1)．


写真1 iLabs SIP Phone相互接続

展示における全体的な印象は，モバイル・オフィスとスモール・オフィスであり，先の三つのキーワードもそれを意識したものに感じられた．

ワイヤレスLAN

展示会場そのものも含め，WLANが浸透していることが感じられた．会場のどこでもIEEE802.11a/b/gでアクセスが可能なのである．アクセス・ポイント(AP)の出展も多く見うけられたが，ここに二つの流れを見つけることができた．WLANが多くなると問題となるのが電波リソースの問題である．この問題に対する解決手段の一つは，1台のAPが，クライアントから見て複数台存在するように見せるというものである．具体的には複数のSSIDを同時に設定可能で，それぞれのSSIDにVLANを割り当てる(Multiple-SSIDという)というもので，3COM(写真2)などが出展していた．なお，3COMのAPにおけるSSIDの数はIEEE802.11g用で32，a/g用ではそれぞれに32の計64SSIDを同時に設定できるというものであった．こうすれば一つのチャネルで，見かけ上複数のAPが存在するようにできる．もう一つの手段は，WLAN-SWを中央に配置し，APのような形状のアンテナ(MACなどはもっていない)を分散配置し，見かけ上APをアンテナとみなしたダイバーシティを行うことにより「面」としてサービスエリアを提供するというもので，AirFlow Networks(写真3)がそうであった．


写真2 3COMのM-SSID AP


写真3 AirFlowのAP

二つの手段の守備範囲としては，前者は不特定多数のサービスが共存しなければならないキャリアによるサービスの提供，後者は，同じ条件のクライアントが多く存在するエンタープライズに向いていると思われる．

VoIP

VoIPに関しては，電話機型は出そろった感がある．新たな製品系列は無線VoIP端末であり，中には待ち受け40時間というものもあった．規格関連はどこも似たようなもので，ワイヤレス部分がIEEE802.11bで，使用するCODECはITU-T G.711(中にはG.729もある)であった．固定型の電話機では呼制御のプロトコルはほとんどがSIPであったが，WLAN PhoneではH.323が多いように感じられた．しかし，近くSIPにするという回答も多かった(写真4)．


写真4 WLAN Phone

ただIEEE802.11側におけるQoS関連の勧告(IEEE802.11e)がまだ完全に決まっていないため，市場が活発になるのは本年後半からであろうか．なお，このようなサービスを見据えたゲートウェイ機器は出そろい始めていた．

セキュリティ

セキュリティは大きく三つの流れがあった．一つはモバイル・オフィスを意識したものであり，SSL VPN関連の機器が見うけられた．もう一つは在宅での使用を想定しているらしい，e-Mailに関連する機能を充実させたファイアウォールであった．要はSPAMとウィルス対策の充実である．三つ目としては，本人の認証に関する製品で，特に目に付いたのが，Memoryexpertsの製品で，USBフラッシュ・メモリに指紋認証を付けたものである(写真5)．これはメモリ空間を二つに分け，だれでもアクセスできる部分と，指紋認証で一致しなければアクセスできない部分を同時に持つことが可能となっていた．


写真5 指紋認証付きUSBメモリ

そのほか

ほかには，今まで米国ではほとんど無視されていたIPv6が目立ち始めたことと，なぜかKVMoIP(Keyboard，Video，Mouse over IP)系の製品が復活していた．トラフィック・マネージメント系の製品も多く見うけられた．

蛇足だが，アジア系(日本，韓国，中国)の来場者のマナーの悪さも見うけられた．特に相手のつごうも聞かずに写真を撮りまくるのはいかがなものかと思う．

ちなみに，来年はLas Vegas convention centerではなく，ホテルが会場になるそうである．

まつもと・のぶゆき （株)タムラ製作所

Show & News Digest

トロンプロジェクト発足20周年記念講演会

■日時：2004年6月2日（水）
■場所：ホテルラフォーレ東京（東京都品川区）

トロンプロジェクト発足20周年を記念して講演会が開催された．

プロジェクト・リーダの坂村 健氏による講演「トロンプロジェクトの20年」では，同プロジェクトの20年にわたる歴史を振り返る講演が行われた．その中で，1984年に同プロジェクトが発足した当時に提唱していた「あらゆるモノにコンピュータを」のコンセプトは，現在は「どこでもコンピュータ＝ユビキタス・コンピューティング」へと継承されていることを挙げ，この将来ビジョンをめざしながら，その第一歩として当時日本の産業界で必要になりはじめていたRTOSの標準仕様提供を目標にしたとのことだった．また，BTRON，TRONキーボード，TRONチップなど，TRONの各種成果物を振り返る内容だった．

さらにそれに留まらず，現在，同プロジェクトが推進しているT-Engineについても紹介された．その中で，T-Kernelが採用しているT-Licenseを取り上げ，他人の知的所有権や特許を侵す危険性を極力回避しながらT-Kernelを発展させるために，あえてGPLではなく，独自のライセンスを採用した経緯などを語った．

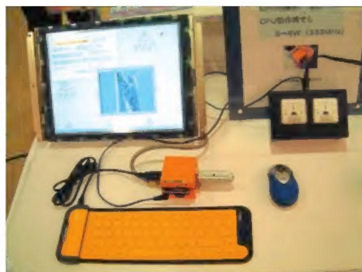
また，併設された展示会場では，各社のT-Engine関連製品が展示されていた．



坂村 健氏



場内のようす



NECエレクトロニクスのブースに展示されたTeacube

DSP&FPGAデザイン・ワークショップ

■日時：2004年5月25日（火）
■場所：パシフィコ横浜（神奈川県横浜市）

CQ出版社の主催によるDSPとFPGAを用いたディジタル信号処理のソフトウェア/ハードウェアへの実装技術に重点を置いたワークショップが開催された．

神戸大学工学部 吉本 雅彦氏による基調講演「動画像圧縮・伸長技術の動向とハード/ソフト実装の課題」は，おもにMPEG-2の動画像圧縮の演算を軽減して，実装可能にする技法を扱った．今後ハイビジョン画質対応などで格段に増加する演算量に対し，動き検出方式の改良により演算量低減をめざすなどの手法が紹介された．

ほかには，来栖川電工（有）井倉 将実氏，小松 浩司氏による「3次元空間を扱うディジタル信号処理系の実装事例」，職業能力開発総合大学校 情報工学科 三上 直樹氏による「DSPの固定小数点処理とC/C++プログラミング」など，本誌執筆者によるセミナも開催され，盛況だった．



セミナのようす

ARM，統合型マルチプロセッサ・コアとDSPコアを発表

■日時：2004年5月26日（水）
■場所：アーバンネット大手町ビル レベル21（東京都千代田区）

ARMは，ARMv6アーキテクチャ・コアを1～4個使用して構成されるマルチプロセッサ・コア「MPCore」を発表した．MPCoreは最大4ウェイのSMP（対称型マルチプロセッシング）およびAMP（非対称型マルチプロセッシング）に対応し，4CPU/550MHz時に最大2600 Dhrystone MIPSの性能を発揮する．これはPentium III Mobile/1GHzに相当するDMIPSとのことだ．各プロセッサはコピーバック・キャッシュをもっているが，これらは修正MESIプロトコルによりキャッシュ内容の整合性が保証される．

また同日，組み込み向けDSPソリューションOptimoDE（オプティモード）も発表された．OptimoDEは，16ビットから256ビットの命令長をもつVLIWアーキテクチャのDSPコアと，DSPをコンフィグレーションするツール・キットをあわせてライセンスするソリューションの名称である．これは，ARMが2003年7月に買収したアデランテ・テクノロジーズ・ベルギー社の技術を元にした製品である．48kHz，320kbpsステレオのMP3データをゲート数30,000個，6MHz動作で再生することができる．

ハッカーの常識的見聞録



今月の常識

RADEON X800に注目しよう

広畑 由紀夫

☆今月は，前回に取り上げたnVIDIA製GeForce6と対をなすATI社製の次世代GPUであるRADEON X800を取り上げます．

● 次世代GPUの必要性

近年，3Dリアルタイム・グラフィックス処理の多くは高精度な3Dを多用したゲームに必須であるばかりではなく，次期Windows――開発コードLonghornにおいてもそのGUI表現などで重要視されています．2D表示に関してはすでに高解像度および表現可能な色の深度などにおいて，一般的な用途ではほぼ十分なものが実現されていると思います．

しかし，筆者はリアルタイムな3D表現においては，まだまだ進歩する必要性があるのではないかと，大規模MMO RPGでもあり，リアルタイム3D処理による表現を重視してもいる「ファイナルファンタジーXI」をプレイしながら日々感じています．

● X800での進化

ATI社製RADEON X800においても，非常に興味深い拡張がなされ，高速化が施されています．具体的には，GDDR3への対応で内部メモリの高速化，パイプラインの拡張による命令セット実行の効率化などが図られています．

X800では，上記の高速化のほかに，AGPバスなどの転送速度の上限を越えたパフォーマンスを引き出せるよう，3Dcノーマル・マップ圧縮技術が使われています．昨今の3D表現では，非常に緻密かつ高速性が要求されるようになり，GPU自身の演算処理能力をより有効に使えるように設計されていますが，現状のAGP8Xバス速度でさえも追いつかないほどのテクスチャが使用されるようになってきています．

PCI Expressバスを搭載した，高速なマザーボードが一般に入手できるようになるまでには，まだ数か月ほど必要だと思われますが，現状でメモリ・バスまでよりよく使うという点において，3Dcノーマル・マップ圧縮技術はたいへん有効なのではないかと考えています．

● 高速な3Dグラフィックスの必要性

コンピュータ・グラフィックスは，その表現力において，一世代前とは格段に進歩してきています．静止画の表現可能色は現状で十分だと思います．また，解像度の面では，DVD鑑賞用途向けなどのワイド表示もありますが，通常の用途では1280×1024ピクセルで十分な表現力をもっていると思います．

筆者がほぼ毎日プレイしている「ファイナルファンタジーXI」では，1600×1280などの，より高解像度なグラフィックス設定が行えるようになっており，使用するコンピュータの性能によっては，より高度なグラフィックス環境でプレイを楽しむことができます．このような環境では，当然ながら1280×1024で表現しているときよりも重い負荷がかかります．

そのような，現状よりも解像度が高くて高度なグラフィックスをリアルタイム処理する性能をもつGPUは，将来的に環境が良くなればなるほど必要になってくるのでしょう．

● 今後の3Dグラフィックス環境へ期待すること

800×600ピクセルの解像度では十分な3D処理が可能で，リアルタイムな表現がタイムラグなく行われていたとしても，1600×1280のような高解像度の設定ではリアルタイム性が損なわれてしまうという現象をよく見かけます．今後のリアルタイム性が必要とされる3Dグラフィックスでは，X800をさらに越える性能を発揮するGPU，そしてそれらを生かすためのバス速度の向上を含めた，システム全体の更新を待たなくてはならないかもしれません．

また，現段階で販売されているGPUが，単体では性能的に限界になっているのであるならば，複数のグラフィックス・ボードによる並列処理を行うといった力業を使うメーカも出てくることでしょう．そのような高度なリアルタイム3Dグラフィックス表現は，3Dゲームのみならず，多様な環境で必要になってくることになるだろうと考えています．

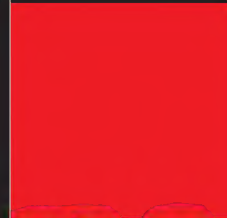
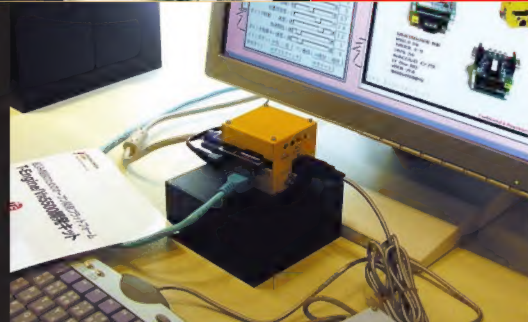
参考URL

(1) ATI社のWebサイト
http://www.ati.com/jp/index.html

(2) ATI社RADEON X800のWebページ
http://www.ati.com/jp/companyinfo/press/2004/4732.html

(3) Intel社ハイ・デフィニション・オーディオのプレス・リリース
http://www.intel.com/jp/intel/pr/press2004/040419.htm

ひろはた・ゆきお　OpenLab.

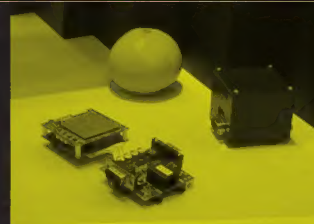
特集　組み込み標準プラットホームをめざして，普及をはかる
新世代TRONアーキテクチャ T-Engine誕生

TRON が注目されている．

すでに組み込み機器では ITRON が幅広く使われており，実際に使った経験のある読者も多いだろう．ITRON のポリシの一つである「弱い標準化」は，統一された仕様の元，多様な実装の自由度が高いものの，ミドルウェアの相互流通性が低いという欠点をもっていた．

そこで T-Engine が提唱された．T-Engine はハードウェア仕様をある程度統一してミドルウェアの流通を促進し，開発プラットホームとしても使えるだけでなく，実際の製品に組み込んでも使えるという野心的な規格である．さらにその上で動作する T-Kernel は，ITRON 仕様をベースに MMU によるメモリ保護機能，動的なアプリケーションのロード機能などを組み込んで高機能化したものだ．これにより，ユビキタス社会を支えるインフラとなる強固で柔軟なアプリケーションの開発が可能になる．

本特集では，期待の高まる T-Engine と T-Kernel について，その思想から仕様，実装に至るまでを一挙解説する．


なぜ，T-Engineが必要とされるのか？

プロローグ

リアルタイムOSの現状

猪飼 國夫

OSといえば，もともとはメイン・フレームという大型機の世界の話でした．最近はWindowsやMacOS，Linux，種々のBSDなどPCベースのOSがすぐに思い浮かびますが，アプリケーションの種類や装備されている機器の数からいうと，じつはリアルタイム系のOSがいちばん数が多いと考えられます．

筆者も，1979年にゲーム機を制御するためにリアルタイムOSのようなものを開発して組み込みました．たかがゲームの機械にと思うかもしれませんが，なぜリアルタイムOSを組み込んだかに言及することで，リアルタイムOSを採用する意味を考えてみたいと思います．

リアルタイムOS以前の組み込み機器

● 昔のゲーム機や家電製品の制御プログラムはアセンブラで書いた

1970年代後半に市場に出周り始めたゲーム機は8ビットのCPUを使うものでした．当時，ゲームをOSの制御下で動かすという考え方はあまりなく，動作速度を上げるためにプログラムをアセンブラでゴリゴリ書いて，それでも間に合わない部分はハードウェアでまかなうという手法が用いられていました．

一方，このころは家電機器や自動車などの多くのコンピュータ以外の機器に，プログラムによる制御が採用されるようになった時代です．プログラムで制御することにより，より使いやすく細かい制御が可能となり，日本の家電製品はその信頼性とともに利便性で世界中で大人気になりました．

これらの家電機器の多くは4ビットのマイコンで制御されていました．ここでの「マイコン」はマイクロコントローラの略で，ROMに書き込まれたプログラムで制御されるI/Oポート付きのCPUです．Intelが発売した4ビットのCPU 4004を見て，数多くの4ビットのCPUが開発されたのです．

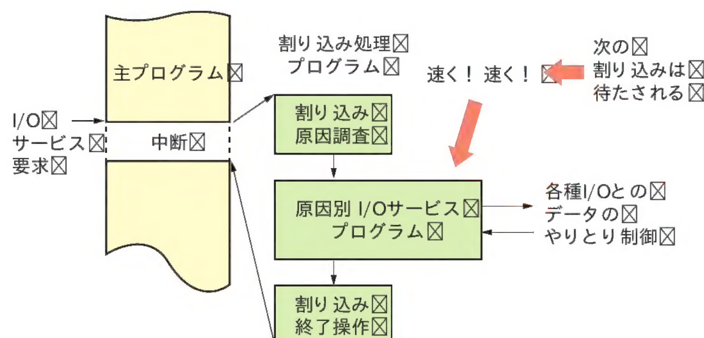


図1　制御を取られっぱなしの割り込み処理での全I/Oサービス方式

● 急ぐサービスは割り込み処理プログラム内で対処

これらの制御プログラムはI/Oからのサービス要求に対して一定時間以内に応えるために，おもに割り込み処理プログラムの中で必要な処理を行うように作られていました（図1）．ものによっては，PLC（Programmable Logic Controller：日本ではシーケンサと呼んでいる）のように，高速にエンドレスで回る構造のプログラムを採用した制御機器もありました．これは昔ながらのリレーによるシーケンス制御をそのままマイコンなどに移植したものです．

家電機器やゲーム機になぜOSが使われなかったかというと，当時OSは（UNIXを別にすれば）IBMなどの大型計算機のスケジュール管理的な動作を行わせるものだという考えがあったからのようです．もちろん，IBM機のOSにはリアルタイム処理の機能が盛り込まれていて，チャネルと呼ばれるI/O（通信回線などリモート機器も含む）では一定時間以内の応答の保証をするプログラムがありました．

リアルタイムOSはなぜ必要か

● メカが主体のゲーム機の制御は割り込み処理での対処はやめた

ゲーム機は4ビットCPUではさすがに能力不足だったので，Z80で制御することにしました．プログラムは当初割り込み処理で対応する予定でしたが，以下のような考えで簡単なリアルタイム処理プログラムを作成することにしました．

1) メカが主体のゲーム機には多くのI/Oがあり，同時に受け付けなければならないサービス要求がある
2) 割り込み処理プログラム内で各I/Oからの要求に対するサービスを行っていたのでは，その処理中にほかの要求に対処するようにプログラムを作ることがたいへんめんどうである
3) ゲーム機が動いている途中で，動きのデータや動作への介入が必要になったとき，別プロセスで動くタスクがあれば，外部から簡単に動作状態のモニタができて機械のデバッグ効率が上がる

ごくごく単純な理由ですが，じつは最大の理由は筆者がリア

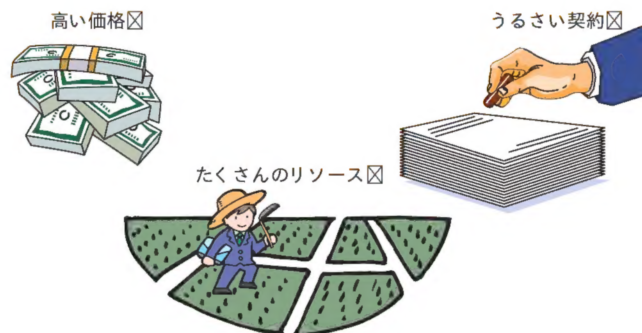


図2 1980年当時の市販リアルタイム OS

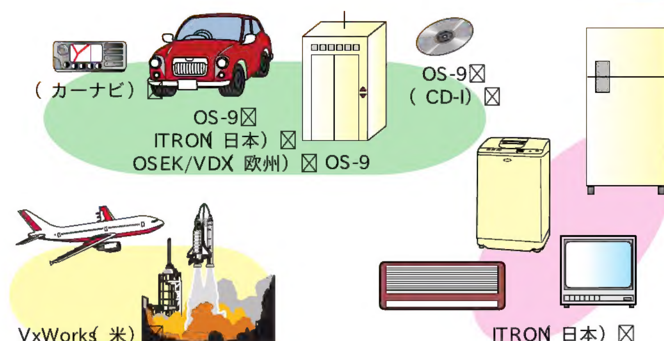


図3 リアルタイム OS の対象別・国別の住み分け

ルタイム OS を作ってみたかったということです．まるで OS オタクのようないいぶんですが，それをもっとまともにして世のためになる構想にしたのが，坂村 健氏の TRON 計画だと思います．

● プロセスが独立して動く便利さを実感した

当時，価格と大きさの点で，使うことができたハードウェアには限界があったため，実際に作成したのはリアルタイム OS の純粋なコア部分だけでした．

8K バイト程度の小さな EPROM に OS と機械制御用プログラム，モニタ用の通信プログラム，当たり管理プログラムを入れなければなりませんでした．

それでもリアルタイム OS の威力は発揮されました．いちばん役に立ったのは，今どきの OS なら常識ですが，ほかのプロセスをモニタ・プログラムから制御(起動・停止・与える条件の変更・内部状況の監視)できたことです．じつは，それまでは Z80 や 8080A を使った制御装置をいろいろ作ってきましたが，リアルタイム OS は使わずに，すべて割り込み処理プログラム内で対処してきました．

● 手が出なかったリアルタイム OS

それまでリアルタイム OS を使わなかった理由は，安価に使える OS としては 1974年に Digital Research 社から出された CP/M しかなく，市販のリアルタイム OS はライセンス料が高くてとても量産品に組み込むことができなかったうえ，大量のメモリを必要とする高機能仕様だったからです(図2)．

そのあとに出た 16ビット版の Microsoft 社の MS-DOS も形だけは UNIX 類似のコマンドが使えましたが，当然ながらシングル・プロセスの OS でした．

当時開発を手伝っていた，リアルタイム処理を要求する工作機械の制御プログラムは MS-DOS 上で動かしたため，デバッグにはたいへん苦労しました．この制御に MTOS というリアルタイム OS が使えないか検討したことがありましたが，性能要求を満たすとコストが合わなくなり断念しました．

実際にリアルタイム OS の利点は，モニタ・プログラムのタスクを独立に持てるというような低水準のものではなく，数多くの I/O からの ms 以下の水準の高速な処理応答要求に対応できる点に意味があります．そして，ゲーム機に搭載したような簡単な OS ですらたいへんな効果を発揮するところがミソです．

いろいろなリアルタイム OS

●リアルタイム OS は競合の世界

リアルタイム OS は本誌 2004年5月号の特集でも取り上げられているように，いろいろな種類が作られています．その中でも OS-9，VxWorks，OSEK/VDX，ITRON などが比較的よく知られています．そのほか PC 上で動くものを含めると Lynx OS，RT-Linux，QNX，Windows CE などがあります．

良くも悪くも競争が成り立っています．特定の種だけが栄えて，多様化されていた遺伝子の種類が減る状況は，自然界では異常です．予想もしなかった原因での大絶滅の危機が襲ってくる可能性が高くなります．その意味ではリアルタイム OS の世界は健全であるといえます(図3)．

● OS-9

OS-9(Operating System for 6809)は MacOS 9とは違います．もっと歴史が古く，Motorola 社から発売された CPU 6809 を産業用の組み込み制御に使う際のリアルタイム OS として旧 Microware Systems 社から 1981年に発売されました．その後，16ビット CPU 68000が出たため移植され，CD-I の制御に使われた以外は，自動車や空調システム，エレベータ制御など，おもに家電機器以外の世界で使われ続けました．

OS-9はアセンブラで書かれていたのですが，C で書き直された OS-9000は x86(80386以降)や MIPS，PowerPC，SH-3などの CPU にも移植されており，リアルタイム OS の原則を押さえて作られているため，その使いやすさと動作の安定性には定評があります．

ただ，基本構想の時期から四半世紀ほど経っているので，仕様上，いろいろと不満なところもないわけでもありません．仮想記憶が使えない，プログラムをモジュール単位で容易に不正コピーされてしまう，などの点で不満が出ることがありました．

以下の特徴は OS-9/9000ファンの方々が利点として挙げているものから抜粋しました．

1) すべてにモジュール構造を採用

2) ROM化可能
3) カーネル以外のモジュールをダイナミックに追加・削除・更新可能
4) UNIXのような階層的なファイル構造を持ちI/Oをファイルとして処理
5) マルチプロセスによるプリエンプティブ・マルチタスク

OS-9が出た当時はCPUの性能に比べて仕様が重く，爆発的に広がるという状況ではありませんでしたが，日本ではFM-7やX68000などの68系パソコンのOSとしても採用されました．現在でも，OS-9系の組み込みシステムの開発には，これらのパソコンやエミュレータが使われています．

● VxWorks

VxWorksはWind River社が発売しているリアルタイムOSです．1987年に発売されたOSで，開発当初は映像関係の処理を行うリアルタイム制御がターゲットでした．

航空機や火星探索機など軍事・宇宙航空関係の制御に多く使われていて，その高速性と安定度は定評があります．Boeing社の次世代航空機7E7にも採用が決定しています．OS-9やWindows，Linuxのように独立したプロセスでのマルチタスクではなく，ITRONと同じようにスレッドで動いています．

一般にスレッドで処理するほうがプロセスをタスクごとに立てるよりは動作を速くできます．スレッド間のデータのやりとりも簡単ですが，それだけタスクごとの分離が甘くなります．

● OSEK/VDX

このOSは欧州の電機業界と自動車業界が主体となって仕様を策定したものです．車載LANの規格であるCANとの併用を図って，米国企業の支配から脱しようという意図もあります．

車載に力を入れているという意味では汎用のOSとはいえませんが，基本的に下位層のハードウェアから独立したOSで，8ビットから32ビットまでの車載に使われるどのCPUにも搭載可能です．

1995年にOSEKとVDK(もともとは独仏で別の仕様だった)を結合した最初の仕様が公開されました．実装は各OSベンダに任されており，仕様だけという形態はITRONと同じです．その特徴は以下のようであると公表されています．

1) 明確なコストの削減と開発時間の短縮
2) いろいろな会社の制御装置のソフトウェアの質の向上
3) 異なった設計思想の制御装置間に標準化されたインターフェースを提供
4) 車の各所に配置された処理機能(機器)を順次利用することで，追加のハードウェアなしでシステム全体のパフォーマンスが向上
5) OSの個々の実装からは完全に独立した仕様を提供し，実装面の仕様は何も規定されていない

自動車各社の車内部のデータ伝送規約やそこで使われるOSを統一することで分散処理も可能とする目的があるようです．これにより，各社の重複投資を避けて自動車各社の標準機器だけではなく，市場から広く安価に器材を調達できるようにという，経済的な側面も伺われます．



ITRON

● ITRONは仕様であって実装は各社まかせ

トロン協会では去る6月2日に20周年記念行事を催しました．

ITRONは1984年に始まったTRON計画の一環として仕様が策定されました．しかしそれは仕様を提示しただけで，具体的な実装については何も規定していませんでした．それでも特定の会社が開発したOSや，特に米国政府の息がかかったOSを使うと，営業上何かと不利益を被るところが多かった日本の製造業各社は積極的にITRONの利用を進めました．

● ITRONへの期待はまだまだ増大

2002年度にトロン協会がまとめた日本における組み込み用リアルタイムOSの利用状況の調査では，ITRONが表1のように市販・自社製・フリーを含めて約4割のシェアを占めています．この調査には筆者も回答を送ったことがあります．

ITRONの利用率は調査のたびに少しずつ伸びてきており，新規の採用希望の調査では5割を越えています．実装の面でもフリーのITRONとしてのTOPPERSプロジェクトやT-Engine計画のT-Kernelに期待が寄せられているようです．

● ITRONの採用と不満の意見

ITRONは以下の観点から日本の多くの開発者の支持を得てきました．

1) OSの仕様や使用条件が特定の会社の利益や特定の国の政策に縛られない
2) 仕様が公開されているので，既成のものでも自分で手を加えることができる
3) 多くのベンダから種々の実装が提供されるので，取捨選択の余地がある
4) 仕様段階ではライセンス料の支払いが不要である
5) 坂村氏が提唱するユビキタス社会へのフィッティングが期待できる

ただ，米国や欧州では，LinuxやGNUなどを生み出したコ

表1　組み込みシステムにおけるリアルタイムOSの利用動向に関するアンケート調査報告書より(トロン協会・日本システムハウス協会)

OSの種類	割合(%)	OSの種類	割合(%)
市販ITRON	24.9	Windows(CE以外)	3.4
自社製ITRON	10.9	Windows CE	2.8
フリーITRON	2.8	MS-DOS	0.6
ITRON合計	38.6	Microsoft社合計	6.8
RTOS不使用	16.2	OS-9	2.8
Linux	10.9	ほかの市販OS	9
自社独自仕様	7.2	ほかのフリーのOS	1.2
VxWorks	7.2		

ミュニティの土壌が本来あるにもかかわらず，なぜか ITRON はあまり支持されていません．インターフェースの仕様だけで実装については何も語らないというのは，仕様で権利を主張する欧米人の目からみると中途半端と映るのでしょうか．また，アジア人が提唱したことへの偏見もあるのかもしれません．

さらに，コミュニティが生み出した財産もすぐに個人の幸せに結びつけるという狩猟民族的な行動様式が，GNU の Stallman 氏のようにあくまでも仕様の開放をめざす TRON プロジェクトと合わないのかもしれません（図 4）．

同じ意味では，中国を TRON や T-Engine プロジェクトの仲間に引き入れようとする努力も，中国人の一族主義や商業主義の文化との関係を考慮に入れなければならないかもしれません．

トロン協会の調査では ITRON への希望として以下のような項目が挙げられています．

1) ソフトウェア・モジュールの標準化
2) 開発環境とのインターフェースの標準化
3) C++/Java の標準バインド
4) デバイス・ドライバの標準化

ユーザは ITRON がゆるい仕様として規定していないところを求めているようです．これらを解決した ITRON がよいかどうかは不明ですが，その対応が ITRON の将来を決めるともいえるでしょう．

ただ，リアルタイム OS は Windows などのエンド・ユーザが直接触る OS とは異なり，専門家が採用を決定する OS なので，大衆的な皮膚感覚だけでは将来を占うことはできません．あえて占うと，組み込み用のリアルタイム OS の世界は RT-Linux（Embedded Linux）と ITRON/T-Kernel の併用になっていくかもしれません．

期待される ITRON の標準実装と T-Engine 計画

● ITRON の実装

ITRON の実装は多くの商用実装や自社専用実装があります．その中で半導体メーカの富士通，日電，ルネサス，東芝などが自社の CPU 向けに専用の ITRON を出しているほかに，いくつかの専業ベンダもあります．

フリーな実装としては高田 広章氏が進めて来た TOPPERS が有名です．もともとは高田氏が豊橋技術科学大学の教授だったときに始めたものですが，今は TOPPERS プロジェクトとして NPO 法人で運営されています．

実装に当たってカーネルをターゲット CPU に依存する部分と独立な部分を分けて，移植性を高めています．また，開発環境はフリーなものだけで完結するようになっており，Linux や Windows 上で動作シミュレーションが行えます．

● TRON チップと T-Engine 計画

TRON 計画は当初から専用の CPU の開発もめざしてきました．

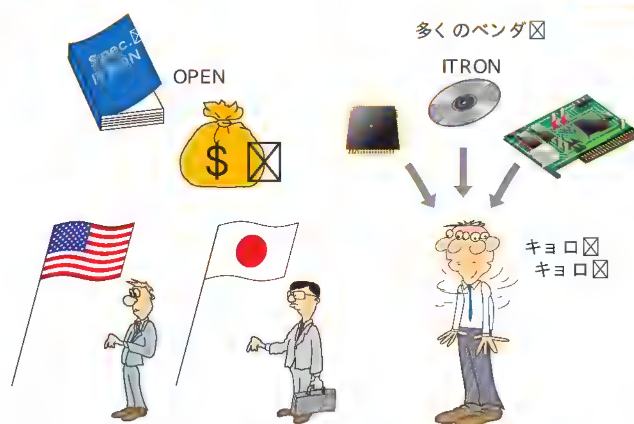


図 4　ITRON が歓迎された理由

すなわち現在の Intel チップと Windows の関係のように，ソフトウェア開発との一体化が目的でした．その方針に乗って 1980 年代後半から 1990 年代初めにかけて G_{MICRO} や M16 などの TRON チップが作られました．筆者もリアルタイム性が強く要求される FA 用途に使おうと考えましたが，チップの供給と開発環境の点であきらめた経緯があります．

しかし，TRON チップの仕様が当時流行した RISC ではなく CISC だったこともあり，M16 などのローエンドの組み込み用チップ以外はあまり採用されませんでした．ただ思想自体は SH など多くの RISC チップにも使われています．

T-Engine 計画では，ハードウェアの内部仕様は厳密に規定していませんが，TRON 計画よりはインターフェースの仕様がかなりはっきりと決めてあります．いくつかのベンダから ARM や SH などの既存のチップをコアにした T-Engine が発表されるなど，G_{MICRO} とは違う動向があり，今後大いなる利用が期待されます．

● T-Kernel

T-Kernel は T-Engine 計画の中で従来の ITRON が担ってきた部分をより強力な標準化で移植性を確保するものです．そのため機種依存部分を T-Monitor などの別の基盤として分離してあります．

詳しい解説は本誌の記事をご覧いただくとして，今後のリアルタイム OS の世界に新しい展開が期待されます．

いかい・くにお　工学博士，（株）エム・アイ・ベンチャー

ITRON から T-Engine へ

第1章 T-Engine の思想

坂村 健

組み込み機器の世界では従来，ITRON が広く使われてきた．コンパクトな仕様で知られ，多数の実装をもつ ITRON は，数多くの機器に組み込まれ，社会を支えている．

ここに新たに T-Engine というプラットホームが投入されようとしている．ハードウェアの仕様を固め，ミドルウェアの流通を容易に行おうとする実践的な試みだ．

そこで本特集の第 1 章として，TRON プロジェクト・リーダの坂村 健氏が，T-Engine のめざすところについて解説する． (編集部)

はじめに

TRON プロジェクトは 1984年にスタートし，今年でちょうど 20周年を迎えた．TRON プロジェクトが最初に手がけた組み込み機器向けのリアルタイム・オペレーティング・システム(リアルタイム OS)である ITRON は，今や，携帯電話をはじめとして，情報機器や家電，車のエンジン制御などに広く利用されている．

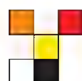
さて，本特集で取り上げる T-Engineと T-Kernel であるが，今までの ITRONとの違いは何かという質問をよく受ける．本特集をもって，T-Engineの考え方を理解していただければと思う．

ご承知のように携帯電話は爆発的に普及し，モデル・チェンジが頻繁に行われ，そのたびに新しい機能やアプリケーションが追加されている．組み込み機器にもかかわらず，そのソフトウェアのステップ数は 100万ステップを越えるという大規模なものだ．開発サイドから見ると非常に短期間で高度なソフトウェアを完成させなければならない．さらに，“ユビキタス・コンピューティング”という新しい応用が生まれつつあり，ソフトウェア開発への要求は減ることはなく増えつづけている．ソフトウェアの生産性をいかに上げるかというのが組み込みシステムに対する最大のテーマである．


写真 1　TRONSHOW 2004のようす

ソフトウェアの生産性を上げるためにはいくつかの方法があるが，一度作ったソフトウェアを再利用していくというのが一番わかりやすい．しかし，現実にはソフトウェアの再利用というのはなかなか難しく，機器内部のハードウェア構成が変わり，モデルが変わるたびに作り直しということが多い．

この問題を解決し，一度作ったソフトウェアは 100年間利用しようということから「100年ソフト」というキャッチ・フレーズを掲げて研究開発を進めているのが T-Engine 開発プラットホームであり，リアルタイム OS の T-Kernel である．



ITRON からの教訓

ITRON を開発したころは，まだマイクロプロセッサが出始めの時期であった．ハードウェア・リソースの規模も小さく，OS はほとんど利用されていない状況だった．ソフトウェアは OS なしで，アプリケーションが I/O から割り込みまでのめんどうを見るように毎回作り込まれていた．

よく使われる処理や割り込み処理，すなわち汎用リアルタイム OS の部分を提供しようというのが ITRON の発想であるが，マイクロプロセッサの能力が低かったため，OS の標準化に際して「弱い標準化」という考えを導入した．32ビット・マイクロプロセッサの領域までを考慮して豊富な機能を用意しつつも，どこまでの機能を用意するかをクラス分けし，マイクロプロセッサのリソースや規模により，クラスを選べるようにした．また，マイクロプロセッサの事情により実装依存を許す部分を設けることにより，多様なマイクロプロセッサに対応できるようにした．その結果，ほとんどのマイクロプロセッサに ITRON が実装され，多くの開発者が ITRON を利用し，組み込み機器向けリアルタイム OSとしてデファクト・スタンダー

ドになった．

ところが「弱い標準」という考え方は，A というマイクロプロセッサ用の ITRON と，B というマイクロプロセッサ用の ITRON は似ているが違う部分もあるということである．PC の世界の Linux にも多くの似て非なるものがあり，その違いによってアプリケーションがうまく動かないということがある．同様に，ITRON の実装による微妙な差は ITRON の上で動くミドルウェアやアプリケーションの作り方に影響し，ほかの ITRON で動かなかったり，移植するためにはむだなコストや時間を負わなければならないということにもなった．

T-Engine の発想

T-Engine の発想は「ミドルウェア流通のプラットホーム」である．このためにリアルタイム OS をシングル・ワン・ソース（ただ一つしかないソース・コード）として提供し，その上で動作するミドルウェアが流通しやすいようにした．もちろん，CPU が異なればバイナリのままでミドルウェアが動くわけではないが，再コンパイルにより利用できるようにしようという考え方である．このシングル・ワン・ソースの OS が T-Kernel である．そして，T-Kernel を動かすための標準ハードウェア・プラットホームが T-Engine である．

T-Kernel はミドルウェアの流通を考え，従来の ITRON が持つスタティック・メモリ・アロケーションに，ダイナミック・メモリ・アロケーションを加え，タスクやセマフォなどのリソース ID を自動で割り当てできるようにしたものだ．さらにファイルやプロセス管理をサポートする上位層として T-Kernel Extension を用意した．現在，T-Kernel 用のミドルウェアの流通を促進するために，T-Engine フォーラムでミドルウェアをデータベースに登録し，eTRON と呼ぶセキュリティ・アーキテクチャを利用して，ミドルウェア・ベンダとミドルウェアを利用するシステム・メーカの間で，電子的にミドルウェアを流通させる T-Dist と呼ぶしくみを整備しつつある．このようなしくみを用意することによって，システム開発者は，自分の欲しいミドルウェアをネットワークで探し，即座に試し，良ければ購入するというようなことができるようになる．

T-Engine と T-Kernel による開発

T-Engine であれば T-Kernel は必ず動作する．しかし T-Kernel を動かすために必ず T-Engine が必要というわけではない．T-Engine と T-Kernel による開発は次のような流れになる．

T-Engine はすでに主要な組み込み用の CPU を搭載したものが何種類も用意されている．システム開発者は，この中から目的に合いそうな CPU を搭載した T-Engine を入手し，T-Kernel や開発環境を整える．この時点でソフトウェア開発がスタートする．そして，ハードウェア・チームは，T-Engine のハードウェアにない機能や，速度が必要とされるためハードウェア化したほうが良いような機能を FPGA などを使って拡張ボードとして作り，T-Engine ボードと組み合わせて機能的には最終目的に近いシステムのハードウェア・プラットホームを完成させる．そしてこれをソフトウェア・チームに引き渡す．



写真 2　TRONSHOW 2004 で展示された T-Engine

ソフトウェア・チームは，拡張ボードのついた T-Engine を使い，利用できそうな流通ミドルウェアも組み合わせ，ソフトウェアの開発を進行させる．その間にハードウェア・チームはハードウェアの一部を ASIC 化したり，ターゲット向けの基板を作り上げたりする．そして，並行して開発しているソフトウェアを搭載して最終製品にしていく．

T-Engine を利用することにより，最初の開発ハードウェア・プラットホームをつくりあげるまでの時間とコストを下げることができる．また，ソフトウェアの開発途中でも，T-Engine を利用してデモンストレーションを行うことができ，マーケティングなどの意見をフィードバックするのに大いに役立つ．従来の組み込み機器用の開発ボードは大型のものが多かった．何度も繰り返すが，T-Engine のハードウェアはあくまで開発のためのプラットホームである．しかし，コンパクトに作ることにこだわったため，開発途中でも製品イメージにできるだけ近いものが作れるようになった．

T-License

T-Kernel はすでに T-Engine フォーラム（http://www.t-engine.org/）よりソースを配布している．T-Kernel のソースの配布を受けるには T-License と呼ぶ契約を締結してもらえば，その条件でだれでも（T-Engine フォーラム会員でなくても）無償で組み込み機器の製品に利用することができる．

ソースをオープンにしているソフトウェアはいくつかあるが，それぞれ利用するための条件が異なる．たとえば GPL では，バイナリを配布した人に対して，改変したソース自体を公

COLUMN

T-License

T-Kernelの配布に際してはT-Engineフォーラムにより定められたT-Licenseが適用される．このライセンスを以下に挙げる．

T-Kernelライセンス

T-License ［T-Kernelのソースコードのライセンス契約］

T-Engineフォーラム

第1条 規定範囲

1.本ライセンス契約は、T-Engineフォーラムより配布を行うT-Kernelのソースコード及びその派生物に関する、著作権ならびに利用条件を定める。

第2条 用語定義

1「T-Kernel」とは、著作権者よりT-Engineフォーラムが委託を受け管理および配布を行うT-Engine用リアルタイムオペレーティングシステムをいう。

2「本ソースコード」とは、T-Kernelのソースプログラム（付随するコメント、ドキュメンテーションを含む）をいう。

3「単純移植されたソースコード」とは、T-Engineフォーラムに登録されたT-Engineハードウェアで稼働するように、T-Engineフォーラム発行のT-Engineハードウェア仕様書に準拠して製作され、本ソースコードのハードウェア依存部のみを改変したものをいう。このうち、T-Engineフォーラムに登録されたものは「本ソースコード」に含める。

4「改変されたソースコード」とは本ソースコードを性能強化、機能追加・削減などを目的として改変して生成されたソースコードをいう。「単純移植されたソースコード」は「改変されたソースコード」の定義には含まれない。

5「バイナリコード」とは本ソースコードまたは改変されたソースコードの全部もしくは一部を含むプログラムをコンパイルして生成された実行形式のコードをいう。

6「本ソースコードの派生物」とは、改変されたソースコードとバイナリコードを総称したものをいう。

7「組み込み製品」とは、本ソースコード、改変されたソースコード、またはバイナリコードを利用し、ハードウェアに実行形式のコードを搭載して作動する機器をいう。

8「最終利用者」とは組み込み製品を購入して使う一般の消費者をいう。

9「システム開発者」とは、組み込み製品を自らまたは第三者に委託して開発し、最終利用者に組み込み製品を有償、無償を問わず提供する者をいう。

10「改変版配布者」とは、改変されたソースコードを製作し、有償無償を問わず第三者に配布する者をいう。

11「改変版パッチ」とは、本ソースコードから改変されたソースコードあるいはそのバイナリコードを生成するための差分あるいはその生成プログラム、生成システム等をいう。

12「パッチ処理の代行」とは、本ソースコードに対して改変版パッチを利用して改変されたソースコードあるいはそのバイナリコードを生成する作業を代行することをいう。

13「ソースコード利用者」とは本ソースコードを利用する者をいう。

14「配布」とは、次のことをいう。

1.インターネット等の通信、放送等により、著作物を特定多数の人に送信すること。

2.インターネット等の通信、放送等により、著作物の送信を不特定の人からの求めに応じ自動的に行うこと。

3.著作物の複製物を、不特定または特定多数の人に頒布すること。

第3条 本ソースコードの著作権

1.本ソースコードの著作権は坂村健が有する。

第4条 利用許諾

1.T-Engineフォーラムは、T-Engineフォーラムの定める所定の登録手続を済ませ、かつT-Licenseに同意した者に対して、以下に定める通り、本ソースコードを無償で利用許諾し、提供する。

2.本ソースコードはT-Engineフォーラムのみより配布を行う。本ソースコードを入手した者は本ソースコードを再配布してはならない。

開しなければならないというような条件がある．組み込みシステムを考えた場合，改変した部分というのはノウハウに属していて公開したくないとうことが一般的である．T-Kernelの配布のためのライセンスは，組み込み機器での利用に適するようにしたT-Licenseと呼ぶ契約となっている（コラムを参照）．

T-Licenseでは，ソースを自由に改変してそれを機器にバイナリの形で組み込んで製品として販売することができ，そのためのライセンス費用も無償である．

おわりに

T-Engineはユビキタス・コンピューティングの世紀を目前として組み込み機器の開発をいかに効率よくするかをテーマに開発，標準化を行ってきた．当初22社の賛同で始まった研究開発の中心的組織であるT-Engineフォーラムも，すでに海外国内あわせて410社という会員の参加を得ている．シングル・ワン・ソース，ミドルウェアの電子流通メカニズム，組み込みシステムに適したT-Licenseなど，次世代組み込み機器の生産性向上にT-Engineが大きな貢献をできることを信じている．

さかむら・けん　東京大学教授，TRONプロジェクト・リーダー

Information ―― NVIDIA，PCI ExpressでIntelと提携
米NVIDIA社のPCI Express搭載のGPU「GeForce PCXファミリ」が，米Intel社のPCI Express対応チップセット発売キャンペーンの一環として，販売やマーケティング，プロモーションに使用されることになった．

3.ソースコード利用者は、次のことをすることができる。

1.第1項により入手した本ソースコードを、自らの研究、開発などの目的のために複製し、改変すること。

2.第1項により入手した本ソースコードを、自らの研究、開発などの目的のために動作させること。

3.第1号により改変した本ソースコードを、自らの研究、開発などの目的のために動作させること。

4.システム開発者は、次のことをすることができる。

1.前項に定める行為。

2.バイナリコードを含む組み込み製品を開発し、製造し、有償無償を問わず最終利用者にこれを提供し、最終利用者に組み込み製品上でバイナリコードを利用させること。

5.ソースコード利用者、システム開発者は本ソースコードあるいはバイナリコードの利用あるいは最終利用者に組み込み製品上でバイナリコードを利用せしめるに際し、別途T-Engineフォーラムの定める方法により本ソースコードを利用した旨を表示する義務を負う。

第5条 改変されたソースコードの配布

1.T-EngineフォーラムのA会員は、所定の登録手続を経て承認されることにより改変版配布者となることができ、当該登録手続を行なったA会員の当該部署に限り、以下に定める方法により、改変されたソースコードの配布をA会員である期間中行うことができる。

2.T-Engineフォーラムは改変版配布者に本ソースコードを提供し、改変版配布者は本ソースコードをもとに改変を行い改変されたソースコード(以下当該「改変されたソースコード」という)を作成し、または本ソースコードから当該「改変されたソースコード」への改変版パッチを作成することができる。

3.改変版配布者は、配布にあたって予め、当該「改変されたソースコード」の名称と概要をT-Engineフォーラムに所定の方法で通知し、登録するものとする。

4.当該「改変されたソースコード」の名称はT-Engineフォーラムの別途定める規定に則るものとし、また規定による表示を改変版パッチならびに当該「改変されたソースコード」に対して行う。

5.改変版配布者は、当該「改変されたソースコード」を、有償無償を問わず第三者に配布することができる。ただし、当該「改変されたソースコード」を入手した者に当該「改変されたソースコード」を再配布させてはならない。

6.改変版配布者は改変版パッチをシステム開発者に提供することができる。ただし、改変版配布者は、提供にあたって、当該システム開発者が第4条1項による正当なソースコード利用者であることを確認する義務を負う。

7.改変版配布者はパッチ処理の代行をシステム開発者に対して行うことができる。

8.システム開発者は、改変されたソースコードをさらに改変して、これをソフトウェア単体で配布することはできない。

9.システム開発者は改変版配布者より配布された改変版パッチあるいはパッチ処理の代行により入手した本ソースコードの派生物を利用して第4条5項と同様の条件にて組み込み製品を最終利用者に利用させることができる。

第6条 単純移植されたソースコードの登録

1.T-Engineフォーラムの会員は新たなT-Engineハードウェアに対し単純移植されたソースコードをT-EngineフォーラムにオリジナルのT-Kernelソースコードとして登録され、配布されるように依頼することができる。

2.前項の配布を依頼する会員は以下の条件を満たすものとする。

1.対象ハードウェアはT-Engine仕様に基づくものであること。

2.依頼にあたって、対象ハードウェアおよびT-Kernelの動作環境をT-Engineフォーラムに一式無償貸与する

3.依頼にあたって、単純移植されたソースコードのテスト結果を添付する

3.T-Engineフォーラムは第1項の依頼を正当なものと認定した場合、当該「単純移植されたソースコード」をT-Kernelソースコードとして登録し配布を行う。但し、本条により登録されるまでの間、単純移植されたソースコードの利用に関して、改変されたソースコードと同様の扱いが認められる。

4.本条により登録された単純移植されたソースコードについて、第3条が適用される。

第7条 周辺ビジネス

1.本ソースコードまたは改変されたソースコードについて、第4条及び第5条に定めた以外の利用を行なう場合には、あらかじめT-Engineフォーラムの許諾を必要とする。

第8条 保証

1.T-Engineフォーラムおよび本ソースコードの著作権者は、本ソースコードが第三者の著作権を侵害していないことを保証する。

2.T-Engineフォーラムおよび本ソースコードの著作権者は、本ソースコードがソースコード利用者の目的に適合することを保証するものではない。

3.T-Engineフォーラムおよび本ソースコードの著作権者は、本ソースコードが第三者の工業所有権を侵害していないことを保証するものではない。また、ソースコード利用者と第三者との工業所有権に関する紛争に関して一切の責任を負わない。

第9条 本ライセンス契約違反に対する措置

1.T-Engineフォーラムは、T-Engineフォーラムの会員であるかどうかを問わず、本ライセンス違反した者に対し、違反の是正と著作権侵害に基づく措置を取るものとする。

2.本ライセンス違反したT-Engineフォーラムの会員は、違反の態様に応じて、T-Engineフォーラムからの退会その他の処分を受ける。

3.本ライセンス契約に関して訴訟の必要が生じた場合には、東京地方裁判所を専属的合意管轄裁判所とする。

第10条 準拠法と言語

1.本ライセンス契約は日本法により支配され、解釈される。

2.本ライセンス契約は日本文および英文で作成される。但し、本ライセンス契約の解釈では日本文が優先する。

新たな組み込みシステム用の開発プラットホーム

第2章 T-Engine ハードウェアの概要

早川 幹/小林 真輔/加藤 淳

T-Engine が従来の開発プラットホームと大きく違う点は，各種の CPU を採用することができるにもかかわらず，その上では同一の OS が動作する点，たんなる開発プラットホームとしてだけではなく，最終出荷製品に組み込んで使うことも考慮してあるということである．

本章では，T-Engine の目的と，そのハードウェアの概要について解説を行う．

（編集部）

T-Engineとは，組み込みシステム用の開発プラットホームの標準化された規格です．T-Engineではハードウェアとリアルタイム OS を標準化することによって，異なるハードウェアに実装する場合においても，OS 上で動作するミドルウェアを再利用することができるようになります．ミドルウェアを再利用することにより，組み込みシステムの生産性が向上し，また，信頼性の高いシステムを実現することができるようになります．

そこで本稿では，なぜ T-Engine を開発するにいたったか，T-Engine プロジェクトでは何をめざし，どのようなことを実現していくのかを説明していきます．それと同時に T-Engine，μT-Engine の規格の概要について述べます．そして，T-Engine の応用システムの一つ，「ユビキタスコミュニケータ」を紹介します．



なぜ今 T-Engine なのか？

ここではなぜ T-Engine なのかという点について述べていきます．これまで TRON プロジェクトでは，組み込みシステム用のリアルタイム OS として，ITRON を研究開発してきました．ITRON はこれまで，携帯電話や自動車，そのほかの多くの家電製品に用いられてきました．その実績を考慮すると，「なぜ ITRON を推進していくのではなく，新たに T-Engine という規格をつくる必要があるのか？」という疑問が湧いてくることと思います．まず，その点について次に述べていきます．

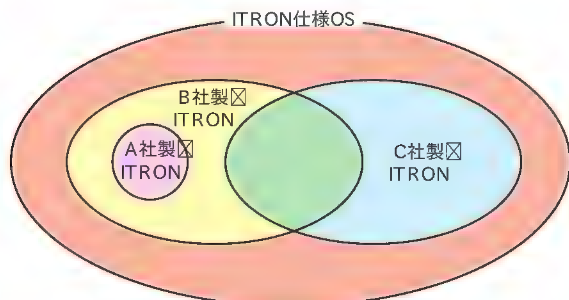


図1　ITRON 仕様 API のサブセットでも“ITRON 準拠 OS”に

● ITRON の教訓

ITRON の特徴の一つとして「弱い標準化」という点があります．「弱い標準化」とは，リアルタイム OS の API など必要最低限の仕様だけを決めて，それ以上の実装上の規定を設けないということです．また，API のサブセット化も許しており，すべての API を実装していなくても「ITRON 準拠 OS」と呼ぶことができました（図1）．

このような方針を採用していたのは，次のような時代背景によります――ITRON の仕様策定は，TRON プロジェクトが始まった 1984 年ごろから始められました．そのため，1984 年ごろのハードウェアを念頭において仕様が考えられていました．当時のハードウェアの性能は，今現在の性能と比較すると非常に非力なものでした．それゆえに，ITRON の仕様をすべて満たすようなハードウェアを開発する場合，ある程度のコスト高になる可能性がありました．仕様が大きいことが原因でハードウェア・コストが高くなることは極力避けたいということで，ITRON では実装の詳細は規定せず，さらに API のサブセット化も許可しました．このことによって，システムごとに OS を最適化することができるようになり，機器を開発するメーカにとって非常に使いやすい OS となりました．

また，ITRON の方針は組み込みシステムの性質にも非常に合っていました．組み込みシステムは汎用のコンピュータとは異なり，ソフトウェアを単体で販売するということはなく，ソフトウェアは必ずハードウェアとセットで販売されています．したがって，メーカにとっては「ソフトウェアを売る」というよりも「システムを売る」ということになるので，システムの価格が安くなることが非常に重要な点でした．そのため，実装方針によって軽く小さくできる ITRON は魅力的だったといえ

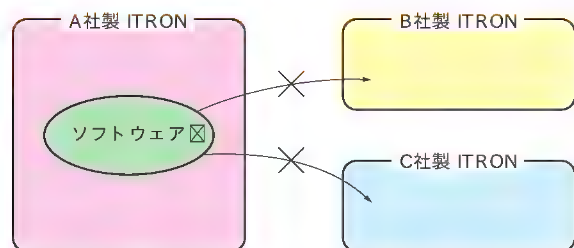


図2　異なる実装のITRON上では正しく動作しないことが多い

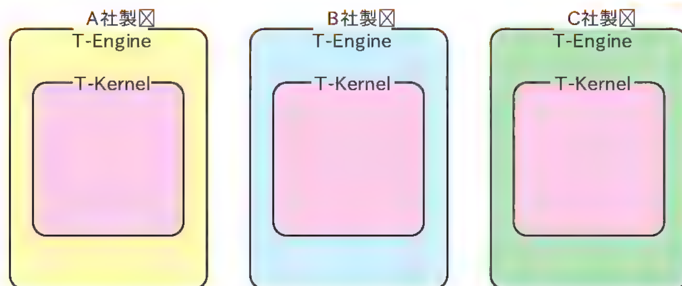


図3　T-Engineの場合，ボード・サイズ，リアルタイムOSの動作はすべて同じ

ます．また，組み込みシステムにはメーカのノウハウが詰まっているため，システムの一部であるソフトウェアを社外に提供するということはありませんでした．そのため，「A社のITRONではどのようなAPIをサポートしていて，どのようなタスク・スケジューリングをしているか」といったOSの仕様にかかわることは，一部の設計者が知るだけで十分でした．それゆえに，実装が異なるITRONが存在しても大きな問題ではありませんでした．

ところが，近年のテクノロジの進歩にともなって半導体の性能が飛躍的に向上してきたことで，前提としてきた状況が一変しました．半導体の性能向上にともなってシステムに高機能化が求められるようになり，その結果，システム開発は大規模化，複雑化してきました．そのため，ソフトウェアの開発にかかる工数も膨大になってきました．このようにソフトウェア開発の工数が膨大になることは，その開発コストがそのまま製品に跳ね返ってしまうことになり，メーカとしては重大な問題となります．したがって，ソフトウェアの開発工数を減らすために今までの開発手法を見直して，改善していく必要性が出てきました．

工数を削減する手法の一つとして，過去に設計したソフトウェアを再利用するという方法があります．過去に設計したソフトウェアを再利用することにより，ソフトウェアそのものを開発する工数を削減できるほか，ソフトウェアをテストする工数も削減することができます．ここで，ITRONを使ったシステムにおいてソフトウェアの再利用を試みた場合，いくつかの問題点がわかってきました（図2）．ITRONはその性質上，仕様の異なるITRONが多数存在します．そのため，仕様が異なるOSの上で動作するソフトウェアをそのまま移植した場合にソフトウェアが動作しない可能性が高いということになります．これは，ある一つの会社内の同じグループ内での再利用を前提とした場合は大きな問題ではありません（実際，これまでも小グループ内での再利用は行われてきた）．しかし，小グループ内での再利用では，今後ますます大規模化・複雑化していくソフトウェアの開発工数を改善することができません．したがって，同じ会社の中での再利用のみならず，さまざまなところで作られたソフトウェアを再利用する枠組みを整備する必要が出てきました．これを実現するのがT-Engineなのです．

● そして「強い標準化」へ

ソフトウェアの再利用を考えた場合にITRONにおいて問題になっていた点は，各社各様のITRONが存在することでした．これはITRONの特徴の一つだったのですが，ソフトウェアの再利用という点では不利であるということは，前述のとおりです．

そこでT-Engineでは，ITRONのようにさまざまな実装が乱立しないように「強い標準化」を行うことで問題を解決することを試みました（図3）．「強い標準化」とは，先ほどの「弱い標準化」とは対照的に，APIなどの仕様だけにとどまらず，実装にまで踏み込んで標準化するということです．

T-Engineでは，リアルタイムOSをシングル・ソース化し，ハードウェアに依存した部分を除いた大部分を共通のCソース・コードにしています．また，ボード・レベルでインターフェース機能を標準化しています．さらに，ボードに搭載されているおもなデバイス・ドライバに関して標準ドライバAPI仕様を定めています．このことにより，標準のAPIを用いることで異なるT-Engineボードにおいてもソース・コードを改変することなしに使用することができます．

T-Engineのめざすところ

この項では，T-Engineのめざすところ，T-Engineのアプローチ，そしてT-Engineのシリーズについて説明していきます．

● T-Engineのめざすところ

T-Engineのめざすところは，ユビキタス・コンピューティングのための標準開発プラットホームです．ユビキタス・コンピューティングは，あらゆるところにコンピュータを配置して，それらをネットワークで接続して動作させるという，これまでにはない概念モデルです．ユビキタス・コンピューティングを実現するシステムを開発する場合，いくつかの問題点に直面します．

まず，一つは生産性・信頼性の問題です．あらゆるところにコンピュータを埋め込む場合には，これまで以上に多種多様な

システムを開発していくことになると思います．こうしたシステムを開発するうえでソフトウェアの生産性・信頼性を向上することが非常に重要になってきます．そのため，前の項目で述べたように「強い標準化」によってミドルウェア流通を実現し，その結果，生産性・信頼性を向上させることを目標としています．

もう一つの問題点は，セキュリティです．あらゆるコンピュータ・ノードがネットワークへ接続される場合には，セキュリティが特に重要な要素となってきます．たとえば，家の中の機器をネットワーク経由で制御するようなシステムを考えた場合に，第三者が不正に侵入して機器を制御してしまう状況になることは非常に危険だといえます（図4）．また，ネットワーク経由であらゆるサービスを提供しようとした場合にサービスを確実に購入者に届けることが必要になりますが，成りすましによって不正にサービスを受けるといった事態は避けたいところです（図5）．このようにセキュリティにまつわる問題は非常に多くあります．そこでセキュリティの問題を解決するためにeTRON（コラムを参照）を開発しています．

このように，生産性・信頼性，セキュリティの問題を解決し，ユビキタス・コンピューティングを実現することがT-Engineの目標とするところです．

● T-Engineのアプローチ

そこでT-Engineでは次のようなアプローチを取ります．まず一つは，オープン・アーキテクチャ，オープン・プラットホームであるということです．オープンであるということは，だれでも開発することができて，だれでも使うことができるということです．これはTRONプロジェクトでいままでとられてきたアプローチであり，技術を普及させるうえで非常に重要なこととなります．

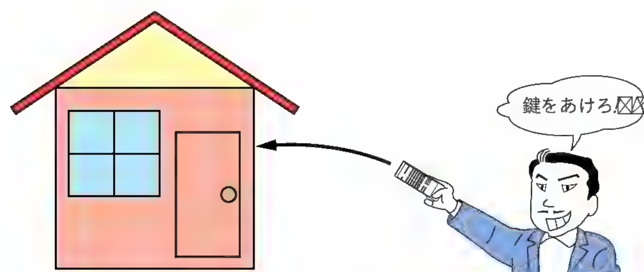


図4　外部よりアクセスして機器を制御

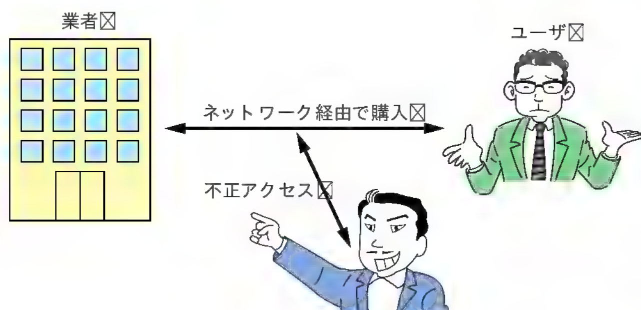


図5　成りすましによる不正アクセス

次にリアルタイムOSやその周辺ソフトウェア，ハードウェアを規格化することです．これは，先ほど述べたソフトウェアの流通を促進するというだけでなく，ハードウェアを規定することで実際の製品イメージに近い開発環境を用いることができるということです．

従来の組み込みシステム開発においては，システム開発のたびに大きな開発用ボードを作成して，システムを開発していました．T-Engineではボードが開発環境として提供され，また大きさもPDAサイズになるので，携帯機器などの実際の製品に近い形での開発が可能となります．

また，eTRON用のインターフェースを標準装備しています．これにより，今後ますます重要になっていくセキュリティの問題を解決し，安全なネットワーク通信を確立することができるようになります．

● T-Engineのシリーズ

T-Engineには，標準T-Engine，μT-Engine，nT-Engine，pT-Engineの4種類があります．

標準T-Engine，μT-Engineは，組み込みシステムの開発プラットホームとして用いることができます．また，少量生産の場合はサイズが許せばそのまま製品に組み込んで利用することができます．nT-Engine，pT-Engineは，ユビキタス・コンピューティングを実現するための実装システムになります．

以降では，それぞれのT-Engineの概要について説明していきます．

▶ 標準T-Engine

標準T-Engine（写真1）は，T-Engineの中でももっとも基本となる組み込みシステム開発環境となります．大きさは大体PDAサイズになります．

タッチ・パネル，USBコネクタ，PCMCIAインターフェースなど標準となっているインターフェースを備えています．

▶ μT-Engine

μT-Engine（写真2）は，標準T-Engineからいくつかのインターフェースを取り除き，サイズをさらに小さくしたT-Engineです．動作しているリアルタイムOSは同じですが，使用できるインターフェースやCPUの性能，メモリなどのハードウェアは，T-Engineと比較して低くなっています．小型のシステムを開発する場合のプラットホームとして使用することができます．

▶ nT-Engine

nT-Engine（写真3）は，センサやコントローラの役割を果たすことができるだけの最低限のインターフェースだけを備えたT-Engineです．センサ・ネットワークを形成することができるネットワーク・インターフェースや機器を操作するためのシリアル出力などのインターフェースがあります．大きさは500円玉程度の大きさとなっています．

COLUMN

eTRON とは？

近年，我々の生活は目まぐるしい発展を遂げています．携帯電話の普及，ADSLに代表されるディジタル加入者線や光ファイバの整備，JR東日本のSuicaなどに見られるRFIDタグの浸透etc….政治，経済，文化といった人間生活のさまざまな活動が効率化され，便利になってきています．そのような中，社会の重要なしくみの一つに「価値情報」というものがあります．

価値情報とは貨幣や証明書，証券，チケットなど日常生活や経済活動にとって欠かせないものです．最近，電子マネーや電子チケットといったことばをよく耳にしますが，今後ますますそれらの価値情報は電子化（ディジタル化）され，社会に流通するようになります．

しかし，ディジタル化された価値情報が，流通する過程で複製されたり改ざんされたりするとどういうことが起こるでしょう？1枚購入した電子チケットがコピーされて2枚になる，証明書の内容が変更できてしまう，電子マネーを自由に増やせてしまう…など，社会を揺るがしかねない事態を招くことは想像に難くありません．

これらの問題に対する一つの答がeTRONです．eTRONの“e”はentity（実体：価値情報）の頭文字を意味しています．eTRONとは，オープンなネットワーク上で電子的な価値情報をセキュアに流通させることができる総合的なアーキテクチャです．

eTRONというセキュリティ・アーキテクチャの重要な要素の一つにeTRONチップがあります．eTRONチップは価値情報を格納するために用いる専用デバイスであり，それ自身が耐タンパ性（複製や改ざんができない性質のこと）を有しています．eTRONチップは内蔵するCPUのビットによりいくつかの種類に分かれ，目的に応じて使い分けることができます．また，インターフェースとして非接触型と接触型が用意されています．

現在，eTRONチップはカード型やSIM型の実装が行われています（写真A）．表Aに eTRONチップの特徴をまとめます．

先程，eTRONチップに価値情報を格納すると説明しましたが，価値情報を安全に操作することを可能にするのがeTP（Entity Transfer Protocol）と呼ばれる専用プロトコルです．eTRONチップとeTPを使用することで，たとえば，携帯電話を用いてチケット販売サイトから安全にチケットを購入したり，携帯電話同士を近付けることでチケットを安全に受け渡しすることが可能になります．

最後に，組み込みシステムにおいても，今後ますますセキュリティが重要になると予想されます．高機能化/小型化/省電力化など，ただでさえ組み込みシステムにおける要求とその開発規模は増加の一途です．T-Engineは身のまわりの至るところ，セキュリティをつねに考慮しなければならない環境に組み込むことを前提としており，eTRONを完全にサポートするように設計されています．つまり，T-Engineにはセキュリティ機能が「デフォルト」でついてきます．ここで，T-Engineをベースに製品開発を行う場合，セキュリティ機能に対する開発工数の大幅な削減が期待できます．

表A　eTRONの種類

種類	接触I/F	非接触I/F	暗号/認証機能
eTRON/8	―	ISO14443 Type-C	秘密鍵
eTRON/16	ISO7816	ISO14443 Type-C	公開鍵
eTRON/32	ISO7816	ISO14443 Type-C	公開鍵



写真A　eTRONチップ



写真1　標準T-Engine

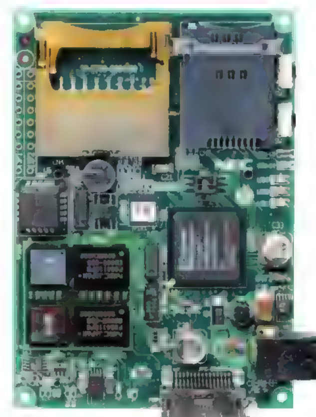


写真2　μT-Engine

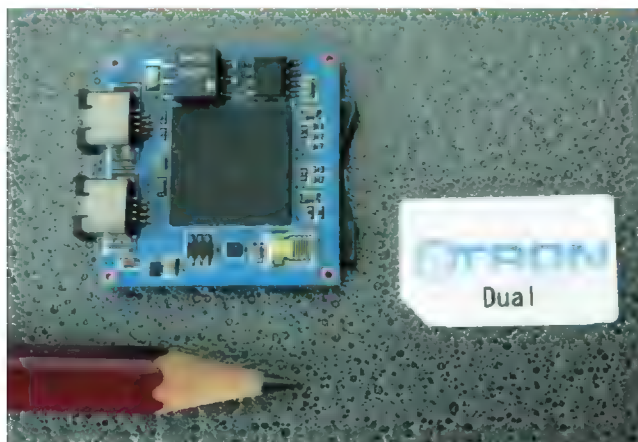


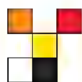
写真3　nT-Engine


写真4　pT-Engine

▶ pT-Engine

pT-Engine(写真4)は，nT-Engineよりもさらに小さくしたT-Engineです．大きさはおよそ1cm角程度の大きさになります．センサを接続するために必要なインターフェース以外は極力排除しています．nT-Engineよりもさらに小型化しているため処理能力は非常に低いのですが，あらゆるところに置くことでユビキタス・コンピューティングを実現することができます．

本稿では特に組み込みシステムの開発プラットホームとして作られている標準T-EngineとμT-Engineについて説明します．

 ワンボード・マイコンとT-Engine

組み込みシステムのプロトタイプ設計には，しばしばワンボード・マイコンが使用されます．

ワンボード・マイコンは，文字どおり“ワンボード”でマイコンを動作させるのに必要な回路が搭載されたボードです．マイコンを中心としてメモリや電源，クロック・ジェネレータなどマイコンを動作させるのに必須な回路を搭載しています．ユーザがプログラムを書いてワンボード・マイコンに転送するだけで，単体で動作させることができます．

ワンボード・マイコンは普通，マイコン上でプログラムを動作させるのに必要最低限の機能しかありません．その代わりに，マイコン上の周辺回路のインターフェースや，ローカル・バスの信号が端子から取り出せるようになっていて，簡単に拡張できるようになっています．これに足りない回路を接続すれば，所望のマイコンを中心としたシステムを構成することが可能です．ワンボード・マイコンはちょっとした試作回路を作るときにとても便利です．大量生産されない組み込み機器では，市販のワンボード・マイコンを最終製品にそのまま使用することも少なくありません．

このようにたいへん便利なワンボード・マイコンですが，いくつか問題があります．

一つ目は，ソフトウェアと拡張ハードウェアの移植性の問題です．たとえば，マイコンをより高機能なものに変える場合を考えてみましょう．ソフトウェアもハードウェアも，共通の部分は再設計が不要であれば，移植工数を大きく削減できます．しかし，「ITRONの教訓」で述べたとおり，同じITRONであっても，実装が異なれば同じプログラムが動作するとは限りません．

これはハードウェアにも同じことがいえます．ワンボード・マイコンの仕様が異なると，周辺の回路も再設計しなければならなくなります．ボード・メーカによって，また使用するマイコンの品種によって，多くのワンボード・マイコンが発売されていますが，ボードの仕様が各社まちまちで統一されていません．CPUを変更したいのでワンボード・マイコンのみを差し替える，といったことは難しくなっています．

二つ目は，使いやすさの問題です．ワンボード・マイコンは，必要最低限の機能しか搭載していない手軽さが利点なのですが，同時に欠点でもあります．開発したプログラムを転送するしくみや，ボタンや液晶などのユーザ・インターフェースも自前で用意しなければなりません．この「自前で用意する部分」が移植性を妨げている原因の一つでもあります．

T-Engineでは，ソフトウェア面においてOSの「強い標準化」によりT-Kernelを規定し，ハードウェア面では周辺機器などの仕様を統一して部品や基板の共通化を図ることにより，移植性の問題を解決しました．T-Kernelは再コンパイルすることでどんなCPUにも対応できる，チップ・フリーのOSです．

さらに，T-Kernel上で動作するミドルウェアについても容易に再利用可能であるように工夫されています．T-Engineフォーラムでは，ミドルウェアが容易に流通可能にするためのミドルウェアの開発規則である「T-Format」を規定しています．

T-Formatでは，ベンダ・コードやC言語のグローバル・シンボル名を規定し，ミドルウェアのソース・コードの流通性を

表1　標準T-Engineと μT-Engineの機能比較

	標準T-Engine	μT-Engine
CPU	32ビット	32ビット
MMU	必須	任意
RAM	必須（容量は任意）	必須（容量は任意）
eTRONカード I/F	SIMコネクタ×1	SIMコネクタ×1
LCDパネルI/F	必須	任意
タッチ・パネルI/F	必須	任意
リアルタイム・クロック	必須	必須
カード I/F	PCMCIA Type II×1	コンパクト・フラッシュ Type II×1 MMCまたはSD×1
USBホスト I/F	USB1.1準拠×1	任意
シリアル・ポート	1チャネル（115.2kbps以上）	1チャネル（115.2kbps以上）
スイッチ類	電源スイッチ リセット・スイッチ NMIスイッチ	電源スイッチ リセット・スイッチ NMIスイッチ 汎用I/O×2
音声入出力	ヘッドホン端子×1 イヤホン・マイク×1	任意
拡張バスI/F	T-Engine規格 1slot	T-Engine規格 1slot
電源コネクタ	EIAJ RC-5320A準拠	EIAJ RC-5320A準拠
ボード・サイズ	75mm×120mm	60mm×85mm

高めています．ベンダ・コードは3文字以上8文字以下の英数字からなるコードで，各ベンダの申請によりT-Engine事務局がユニークであることを確認のうえ，承認することで取得されます．

このベンダ・コードを利用して，C言語のグローバル・シンボル名を以下のように規定することで異なるベンダが提供するプログラムをリンクする際の不具合が防止できます．

▶ミドルウェア名

ベンダ・コード＋“_”＋機能名［＋“_”＋詳細機能名］

詳細機能名は省略可能．

▶exportされる関数名

ベンダ・コード＋“_”＋機能名［＋“_”＋詳細機能名］

＋“_”＋関数識別子

関数識別子は英数字を使った2文字以上の文字列で，その関数の機能名を表す．

▶exportされる大域変数名

ベンダ・コード＋“_”＋機能名［＋“_”＋詳細機能名］

＋“_”＋変数識別子

変数識別子は英数字を使った2文字以上の文字列で，その関数の変数名を表す．

たとえば，ベンダ“unl”により提供された“mpeg2”処理プログラムの基本ヘッダ・ファイル名は，“unl_mpeg2_basic.h”などとなります．

また，プログラムのバイナリ形式としてELF（Executable and Linking Format）仕様を満たすことと定められています．CPUやコンパイラに依存する部分でELFに規定できない部分については，T-Engineフォーラムが別途定めるGNUベースの開発環境の実装に従うことで，バイナリ・コードについても流通性を高めることができます．

また，使いやすさという面では，T-Engineは標準的通信インターフェースやユーザ・インターフェースを備え，高機能であると同時に非常にコンパクトです．PCに接続してのリモート・デバッグはもちろん，メモリ・カードやUSBからブートして単独でのデバッグも可能です．

なお，T-Engineでは，開発環境のリファレンスにGNUのgdbを採用しています．

T-Engine，μT-Engineの規格概要

● ボード構成

開発プラットホームである標準T-EngineとμT-Engineの規格について説明します．標準T-EngineとμT-Engineの機能の比較を表1に示します．

標準T-Engineは，PDA，電子ブック，携帯電話等の携帯情報機器，インターネット情報端末，セット・トップ・ボックス，情報家電のコントローラなどを応用ターゲットとした開発プラットホームです．CPUは32ビットで，MMU（Memory Management Unit：メモリ管理機能）を搭載しています．ユーザ・インターフェースとしてタッチ・パネルつき液晶画面やボタン，また音声の入出力を備えています．また，PCMCIAやUSBホスト機能など汎用インターフェースも豊富です．外形もたいへん小型なので，バッテリを付けてPDAのプロトタイプとしても使用できます．

一方，μT-Engineは，情報家電のコントローラや車載制御機器，ネットワーク応用機器を応用ターゲットとしたリファレンス・ボード，開発プラットホームとして位置づけられています．画面や音声などのユーザ・インターフェースは必須ではありませんが，その分小型です．標準T-Engineより組み込み指向であるといえます．CPUは32ビットですが，MMUは必須ではありません．カード・インターフェースとして，PCMCIAのかわりにコンパクト・フラッシュ・インターフェースとMMCまたはSDカード・スロットがついています．2系統以上用意されているのは，片方をプログラムを格納するストレージとして用いているときに周辺機器とのインターフェースの拡張に使用できるようにするためです．

標準T-Engine，μT-Engineは搭載する周辺インターフェースに相違はありますが，基本的に同一アーキテクチャであり，ともにT-Kernelが動作します．

これまでにリリースされているT-Engine/μT-Engine製品の一覧を表2に示します．SH，M32R，FR400，ARM，MIPSなど世界で広く使われている組み込みプロセッサの多くをサポートしています．

プロローグ 1 2 3 4 5 6 7 APPENDIX APPENDIX

表2 T-Engine製品一覧

	標準T-Engine開発キット						
	T-Engine/SH7727	T-Engine/SH7751R	T-Engine/SH7760	T-Engine/V_R5500	T-Engine/ARM720-S1C	T-Engine/ARM920-MX1	T-Engine/ARM926-MB8
ボード・ベンダ	(株)日立超LSIシステムズ	(株)日立超LSIシステムズ	(株)日立超LSIシステムズ	NECエレクトロニクス(株)	横河ディジタルコンピュータ(株)	横河ディジタルコンピュータ(株)	横河ディジタルコンピュータ(株)
CPUベンダ	(株)ルネサステクノロジ	(株)ルネサステクノロジ	(株)ルネサステクノロジ	NECエレクトロニクス(株)	セイコーエプソン(株)	Motorola, Inc.	富士通(株)
CPUチップ名称	SH7727	SH7751R	SH7760	V_R5500	S1C38000	MC9328MX1	MB87Q1100
CPUコア名称	SH3DSP	SH4R	SH4R	MIPS IV	ARM720T	ARM920T (DragonBall i.MX1)	ARM926EJ-S ARM946EJ-S
CPU動作周波数	96MHz	240MHz	200MHz	400MHz	72MHz	200MHz	200MHz
キャッシュ容量(命令)	16Kバイト(Mixed)	16Kバイト	16Kバイト	32Kバイト	16Kバイト	16Kバイト	16Kバイト×2
キャッシュ容量(データ)	↑	32Kバイト	32Kバイト	32Kバイト	16Kバイト	16Kバイト	16Kバイト×2
MMU	あり	あり	あり	あり	あり	あり	あり
CPU内蔵コプロセッサ	DSP	FPU	FPU	FPU	—	—	—
拡張バスI/F	ローカル・バス	ローカル・バス/PCI	ローカル・バス	ローカル・バス/PCI	ローカル・バス	ローカル・バス	ローカル・バス
オンボード・フラッシュ・メモリ	8Mバイト	8Mバイト	8Mバイト	16Mバイト	8Mバイト	16Mバイト	16Mバイト
オンボードRAM	32Mバイト	64Mバイト	64Mバイト	128Mバイト	32Mバイト	64Mバイト	64Mバイト
備考	—	グラフィック向け行列演算	グラフィック向け行列演算	オンボードPLDにユーザ・ロジック搭載可能	—	—	デュアルプロセッサ構造

	標準T-Engine開発キット			μT-Engine開発キット			
	T-Engine/ARM720-LH7	T-Engine/ARM922-LH7	T-Engine/TX4956	μT-Engine/SH7145	μT-Engine/M32104	μT-Engine/V_R4131	μT-Engine/ARM7-LH79532
ボード・ベンダ	横河ディジタルコンピュータ(株)	横河ディジタルコンピュータ(株)	東芝情報システム(株)	(株)日立超LSIシステムズ	(株)ルネサステクノロジ	NECエレクトロニクス(株)	横河ディジタルコンピュータ(株)
CPUベンダ	シャープ(株)	シャープ(株)	(株)東芝	(株)ルネサステクノロジ	(株)ルネサステクノロジ	NECエレクトロニクス(株)	シャープ(株)
CPUチップ名称	LH79520	LH7A400	TMPR4956CXBG	SH7145	M32104	V_R4131	LH79532
CPUコア名称	ARM720T	ARM922T	TX49/H4	SH2	M32R	MIPSII	ARM7DMI
CPU動作周波数	77.4MHz	200MHz	400MHz	50MHz	216MHz	200MHz	50MHz
キャッシュ容量(プログラム)	8Kバイト(Mixed)	8Kバイト	32Kバイト	—	8Kバイト	16Kバイト	4Kバイト(Mixed)
キャッシュ容量(データ)	↑	8Kバイト	32Kバイト	—	8Kバイト	16Kバイト	↑
MMU	あり	あり	あり	なし	なし	あり	なし
CPU内蔵コプロセッサ	—	—	FPU	乗算・積和演算器	積和演算器	—	—
拡張バスI/F	ローカル・バス	ローカル・バス	ローカル・バス/PCI	ローカル・バス	ローカル・バス	ローカル・バス/PCI	ローカル・バス
オンボード・フラッシュ・メモリ	8Mバイト	8Mバイト	16Mバイト	1Mバイト	4Mバイト	16Mバイト	8Mバイト
オンボードRAM	32Mバイト	32Mバイト	128Mバイト	1Mバイト	16Mバイト	32Mバイト	32Mバイト
備考	—	—	低消費電力CPU搭載	開発中 チップ内蔵フラッシュ・メモリ256Kバイト	—	オンボードPLDにユーザ・ロジック搭載可能	開発中 LCD I/F付き

Information ——プリント基板通販サイト「P板.com」の専用パターン設計CADを無料ダウンロードできるサービスを開始
(株)インフローが運営するプリント基板のインターネット通販サイト「P板.com」(http://www.p-ban.com/)が,(株)ニソールが開発したパターン設計CAD「CADLUS X」の無料ダウンロード・サービスを開始した.

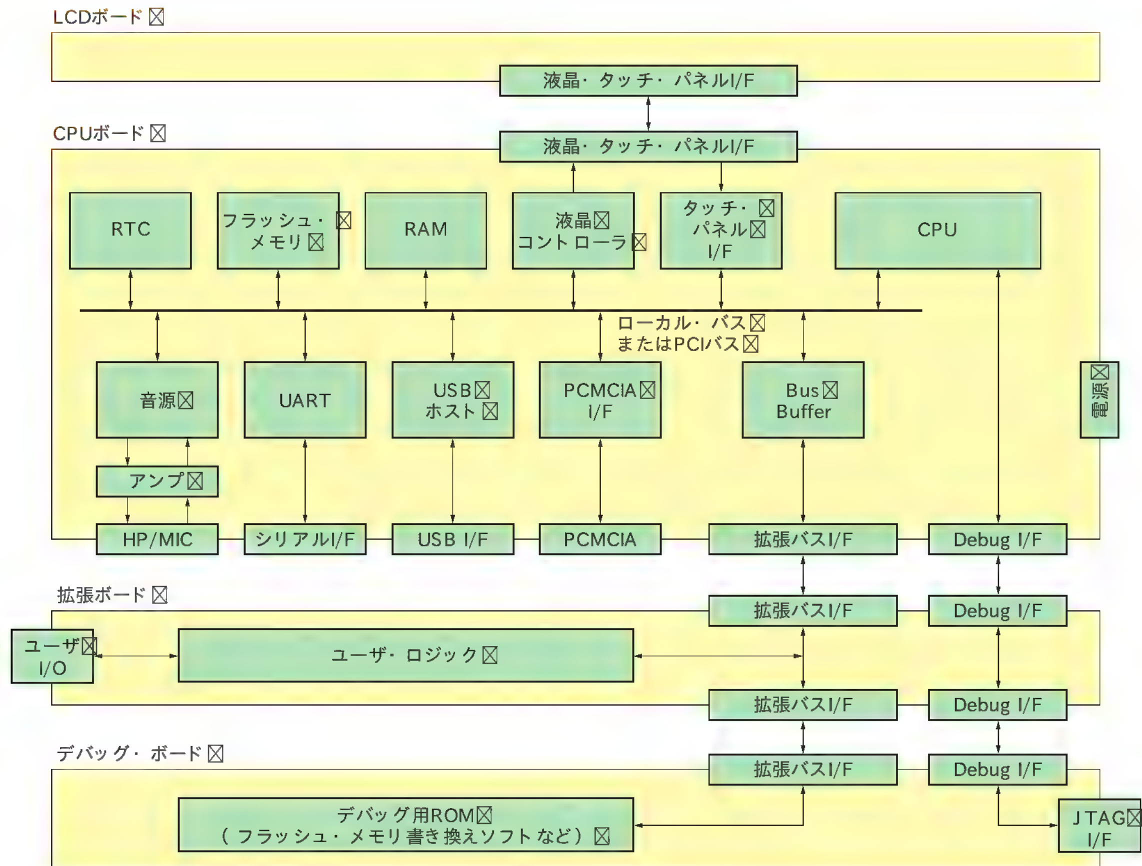


図6　標準T-Engineのシステム構成

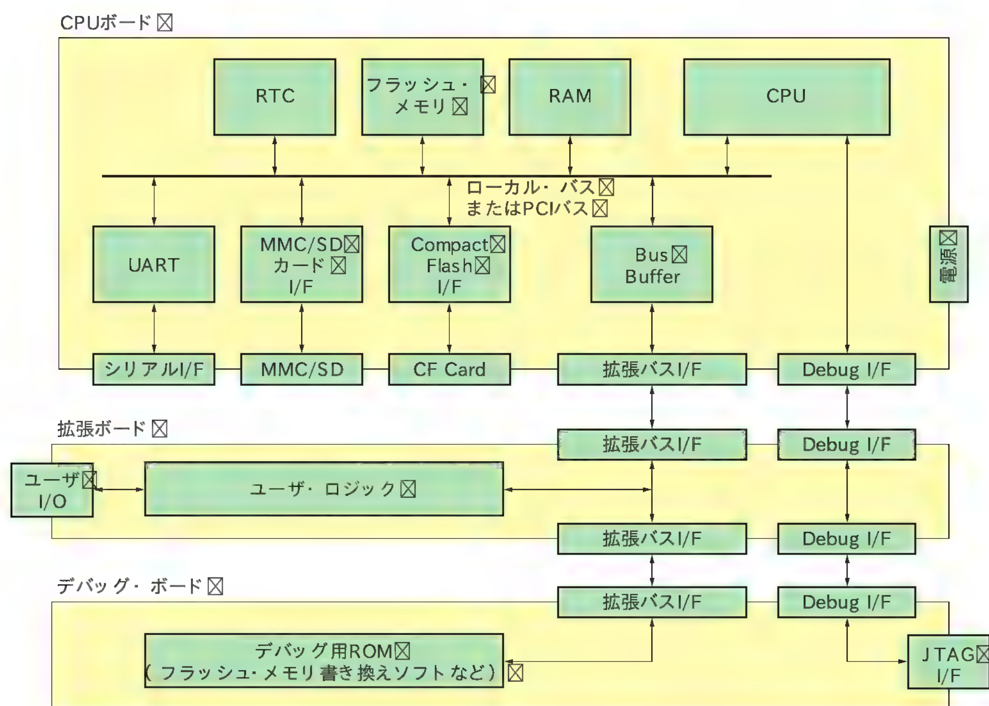


図7　μT-Engineのシステム構成

T-Engineのシステム構成を図6に示します．CPUボードを中心に，LCDボード，拡張ボード，デバッグ・ボードから構成されます．CPUボードは単体での動作が可能であり，不要であればLCDボードや拡張ボード，デバッグ・ボードは取り外しが可能です．

μT-Engineのシステム構成を図7に示します．CPUボードを中心に，拡張ボード，デバッグ・ボードから構成されます．標準T-Engineと違い，LCDボードは必須ではありません．

1）CPUボード

CPUボードには，CPU，メモリ，リアルタイム・クロック

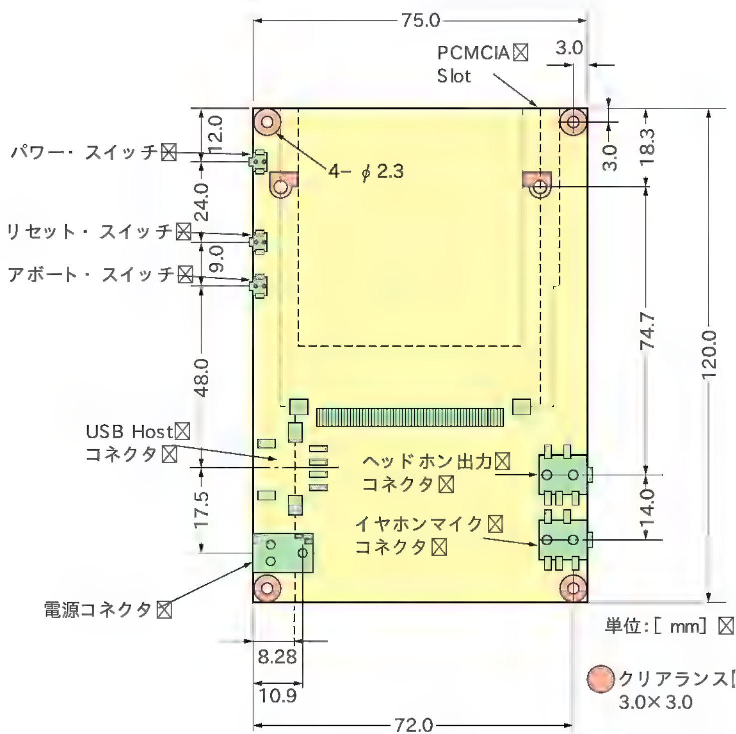


図8　T-Engine CPUボード・コネクタ配置（A面）

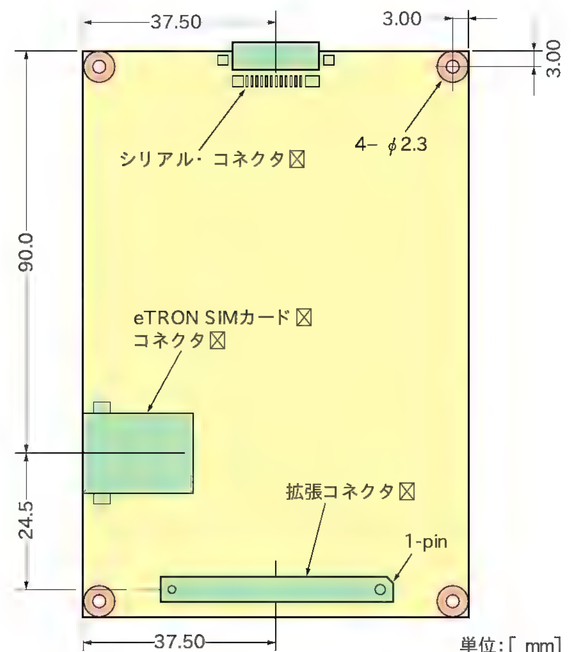


図9　T-Engine CPUボード・コネクタ配置（B面）

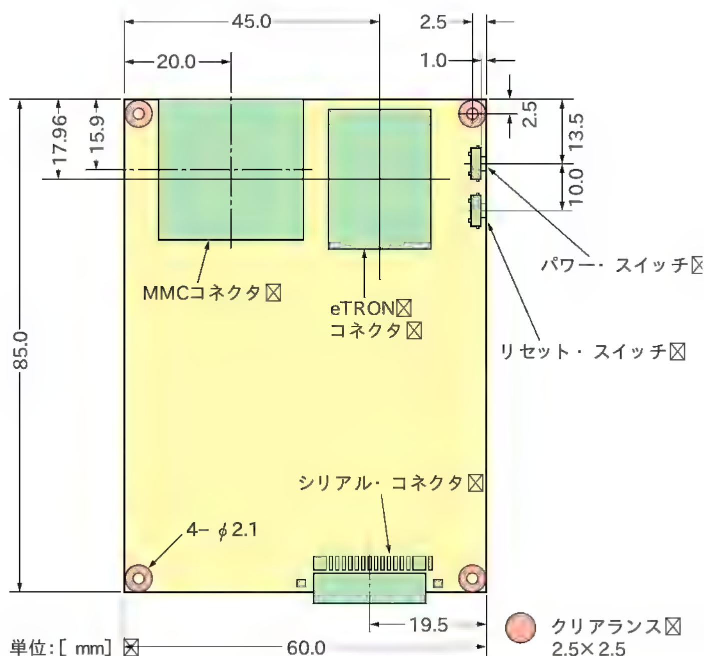


図10　μT-Engine CPUボード・コネクタ配置（A面）

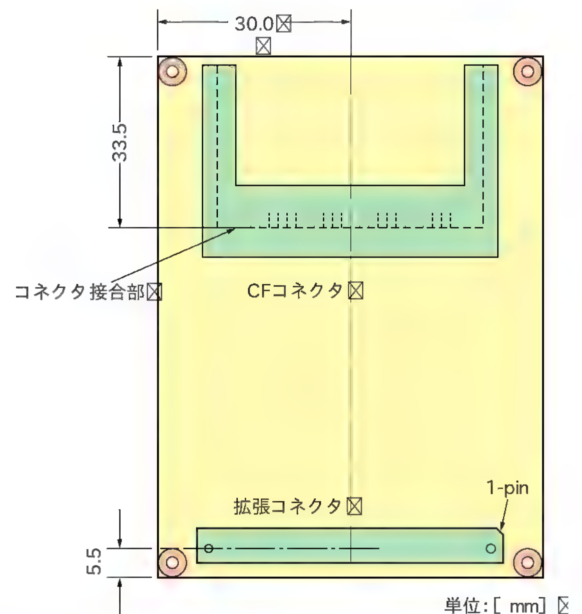


図11　μT-Engine CPUボード・コネクタ配置（B面）

や，PCMCIA や USB ホスト機能などの各種インターフェースが搭載されています．また，セキュリティ・チップであるeTRON を搭載するための SIM カード・スロットがついています．CPU ボードのみで T-Kernel が動作します．

CPU ボードは，外形寸法，各種スイッチおよびコネクタの実装位置が規定されます（図8～図11）．これにより，CPU の違いに係わらず，同じ筐体が使用できます．

2）LCD ボード

LCD ボードは，タッチ・パネル付きの液晶と，ボタンなどのユーザ・インターフェースを備えています．LCD ボードの仕様は任意ですが，現在発売されている T-Engine 向けの LCD ボードには，タッチ・パネル付きの解像度 240×320ドットの TFT カラー液晶が搭載されています．また，ボタンは二つの押しボタン・スイッチのほかに，一つのカーソル・スイッチを備え，方向の入力が可能です．

また，製品により，赤外線受光ユニットを備えます．

3）拡張ボード

拡張コネクタに拡張ボードを接続してインターフェースを拡張したり，ユーザ回路を実装することができます．拡張コネクタは，下面がプラグ，上面がレセプタクルになっていて，図12のようにスタックできるようになっています．スタックしたボードは CPU ボードの下に重ねるように設計しても良いですし，デバッグに便利なようにボードの外側に張り出すように設計してもかまいません．

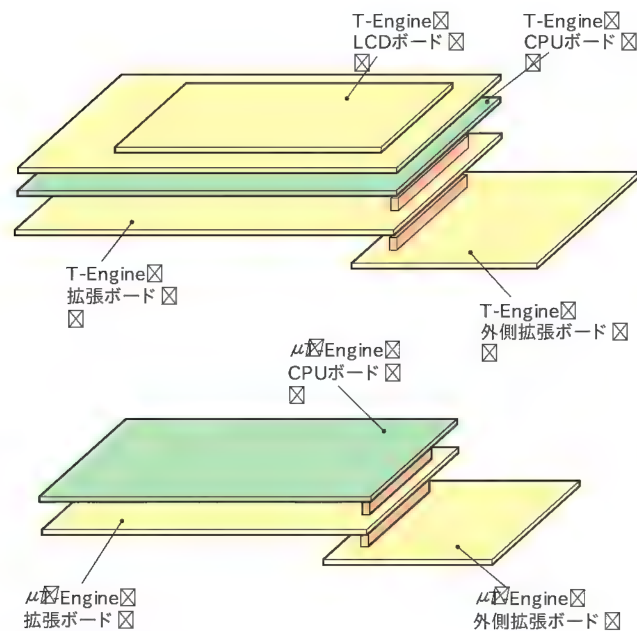
基板の大きさは図13のように規定されています．基板の幅は CPU ボードと同一ですが，長さ方向には 3cm 単位で延長することが可能です．

4）デバッグ・ボード

デバッグ・ボードには，JTAGインターフェースや，CPU ボード上のフラッシュ・メモリを書き換えるためのプログラムを格納した ROM が搭載されています．

● 拡張コネクタの仕様

CPU ボードと拡張ボード間は，140ピンのコネクタによって接続されます．T-Engineフォーラムにより，拡張コネクタの配置と，電源端子のアサインが規定されています．拡張コネクタの配置は，図9，図11，図14を参照してください．電源端子は，140ピンのコネクタのうち，133～136ピンを VBAT（電源端子．3.6～4.3V），137～140ピンを GND（グラウンド）



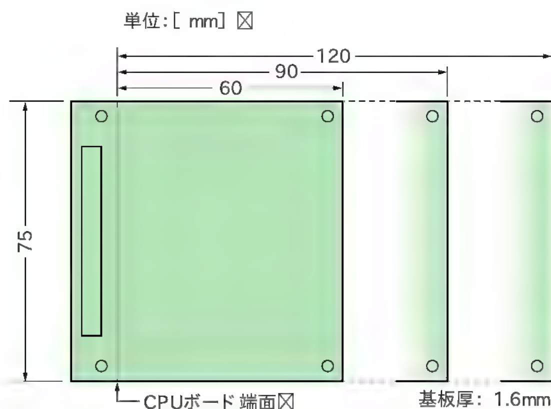


図12　T-Engine，μT-Engine の拡張ボード



図14　140ピン拡張バス・コネクタ

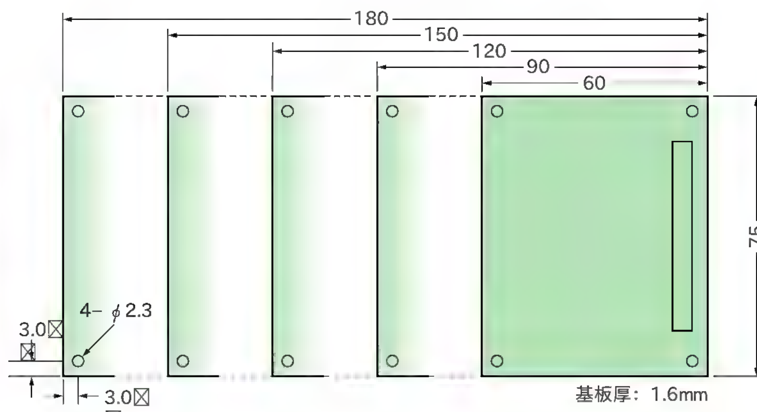


図13　拡張ボードの基板サイズの規格

表3　T-Engine/μT-Engineの拡張コネクタ・アサイン例

ピン	信号名	I/O	ピン	信号名	I/O
1	5V	—	71	A24	O
2	5V	—	72	A25	O
3	5V	—	73	EPROMCE#	O
4	5V	—	74	CS2#	O
5	D0	I/O	75	CS4#	O
6	D1	I/O	76	CS5#	O
7	D2	I/O	77	RDWR	O
8	D3	I/O	78	BS#	O
9	D4	I/O	79	GND	—
10	D5	I/O	80	GND	—
11	D6	I/O	81	RD#	O
12	D7	I/O	82	WAIT#	I
13	D8	I/O	83	WE0#	O
14	D9	I/O	84	WE1#	O
15	D10	I/O	85	WE2#	O
16	D11	I/O	86	WE3#	O
17	D12	I/O	87	GND	—
18	D13	I/O	88	GND	—
19	D14	I/O	89	IRQ0#	I
20	D15	I/O	90	IRQ1#	I
21	GND	—	91	IRQ2#	I
22	GND	—	92	IRQ3#	I
23	D16	I/O	93	NMI_IN	I
24	D17	I/O	94	RST_IN#	I
25	D18	I/O	95	RST_OUT#	O
26	D19	I/O	96	DREQ0#	I
27	D20	I/O	97	DRAK0#	O
28	D21	I/O	98	DACK0#	O
29	D22	I/O	99	ROMSEL	I
30	D23	I/O	100	BASE#	I
31	D24	I/O	101	GND	—
32	D25	I/O	102	GND	—
33	D26	I/O	103	SH7727_TXD2	O
34	D27	I/O	104	SH7727_RXD2	I
35	D28	I/O	105	SH7727_RTS2	O
36	D29	I/O	106	SH7727_CTS2	I
37	D30	I/O	107	—	—
38	D31	I/O	108	ASEMD0	I
39	GND	—	109	GND	—
40	GND	—	110	GND	—
41	CKIO	O	111	TCK	I
42	GND	—	112	TMS	I
43	GND	—	113	TRST#	I
44	GND	—	114	TDI	I
45	A0	O	115	TDO	O
46	A1	O	116	ASEBRKAK#	O
47	A2	O	117	3.3VSB	—
48	A3	O	118	3.3VSB	—
49	A4	O	119	3.3VSB	—
50	A5	O	120	3.3VSB	—
51	A6	O	121	AUDATA0	I/O
52	A7	O	122	AUDATA1	I/O
53	A8	O	123	AUDATA2	I/O
54	A9	O	124	AUDATA3	I/O
55	A10	O	125	AUDSYNC#	O
56	A11	O	126	AUDCK	I
57	A12	O	127	3.3V	—
58	A13	O	128	3.3V	—
59	A14	O	129	3.3V	—
60	A15	O	130	3.3V	—
61	GND	—	131	3.3V	—
62	GND	—	132	3.3V	—
63	A16	O	133	VBAT_IN	—
64	A17	O	134	VBAT_IN	—
65	A18	O	135	VBAT_IN	—
66	A19	O	136	VBAT_IN	—
67	A20	O	137	GND	—
68	A21	O	138	GND	—
69	A22	O	139	GND	—
70	A23	O	140	GND	—

（a）PCIバスなし（SH7727）

ピン	信号名	I/O	ピン	信号名	I/O
1	GND	—	71	AD2	I/O
2	PCLK2	O	72	AD3	I/O
3	GND	—	73	GND	—
4	PCLK1	O	74	AD1	I/O
5	GND	—	75	GND	—
6	PCLK0	O	76	AD0	I/O
7	REQ2#	I	77	PCIRST#	O
8	VCCIO	—	78	LOBAT	—
9	REQ1#	I	79	MPOWER	O
10	VCCIO	—	80	INTA#	I
11	REQ0#	I	81	WAKEUP	I
12	GNT2#	O	82	INTB#	I
13	GND	—	83	INT1#	I
14	GNT1#	O	84	INTC#	I
15	GND	—	85	INT2#	I
16	GNT0#	O	86	INT0#	I
17	AD31	I/O	87	A17	O
18	IDSEL2	O	88	CS1#	O
19	AD30	I/O	89	A16	O
20	IDSEL1	O	90	CS0#	O
21	AD29	I/O	91	A15	O
22	IDSEL0	O	92	IORDY	I
23	AD27	I/O	93	GND	—
24	AD28	I/O	94	A7	O
25	GND	—	95	A14	O
26	AD26	I/O	96	A6	O
27	GND	—	97	A13	O
28	AD25	I/O	98	A5	O
29	AD23	I/O	99	A12	O
30	AD24	I/O	100	A4	O
31	AD22	I/O	101	A11	O
32	CS2#	O	102	A3	O
33	AD21	I/O	103	A10	O
34	EPCE#	O	104	A2	O
35	AD19	I/O	105	A9	O
36	AD20	I/O	106	A1	O
37	GND	—	107	A8	O
38	AD18	I/O	108	A0	O
39	GND	—	109	RESV	—
40	AD17	I/O	110	WR#	O
41	CBE3#	I/O	111	GND	—
42	AD16	I/O	112	RD#	O
43	CBE2#	I/O	113	D15	I/O
44	STOP#	I/O	114	D7	I/O
45	LOCK#	I/O	115	D14	I/O
46	PERR#	I/O	116	D6	I/O
47	IRDY#	I/O	117	D13	I/O
48	TRDY#	I/O	118	D5	I/O
49	GND	—	119	D12	I/O
50	FRAME#	I/O	120	D4	I/O
51	GND	—	121	D11	I/O
52	DEVSEL#	I/O	122	D3	I/O
53	PAR	I/O	123	D10	I/O
54	SERR#	I/O	124	D2	I/O
55	CBE1#	I/O	125	D9	I/O
56	AD15	I/O	126	D1	I/O
57	CBE0#	I/O	127	D8	I/O
58	AD14	I/O	128	D0	I/O
59	AD12	I/O	129	VBAT	—
60	AD13	I/O	130	VBAT	—
61	GND	—	131	VBAT	—
62	AD11	I/O	132	VBAT	—
63	GND	—	133	VBAT	—
64	AD10	I/O	134	VBAT	—
65	AD8	I/O	135	VBAT	—
66	AD9	I/O	136	VBAT	—
67	AD6	I/O	137	GND	—
68	AD7	I/O	138	GND	—
69	AD4	I/O	139	GND	—
70	AD5	I/O	140	BRD_IN#	—

（b）PCIバスあり（SH7751）

表4　シリアル・インターフェースのピン・アサイン

端子番号	信号名	入出力	意　味
1	GND	―	グラウンド
2	TxD	O	送信データ
3	RxD	I	受信データ
4	GND	―	グラウンド
5	RTS	O	送信要求
6	CTS	I	送信可能
7	GND	―	グラウンド
8	Reserved	―	予約
9	Reserved	―	予約
10	Reserved	―	予約
11	Reserved	―	予約
12	Reserved	―	予約
13	Reserved	―	予約
14	Reserved	―	予約
15	Reserved	―	予約

表5　eTRONインターフェースのピン・アサイン

端子番号	信号名	入出力	意　味
1	V_{cc}	―	電源端子
2	Reset	I	リセット
3	Clock	I	クロック
4	Reserved	―	予約
5	GND	―	グラウンド
6	V_{pp}	―	プログラム電圧
7	I/O	I/O	データ入出力
8	Reserved	―	予約

として規定しています．

拡張コネクタには搭載された CPU のバス信号が出力されますが，具体的な信号線の端子配置は実装依存とされています．なぜなら，CPU によってバスのプロトコルや電気的特性が異なるからです．表3に拡張コネクタのピン・アサインの例を示します．拡張コネクタの仕様は PCI バスが出力されているものとされていないものの大きく二つに大別されます．PCI バスを持たない T-Engine どうし，PCI バスを持つ T-Engine どうしは，電源端子の位置等，似たピン配置になっています．ただし，同じ名前・機能の信号線であっても電気的特性やタイミングは CPU により異なる場合があります．ローカル・バスを使用した拡張ボードは，必ずしも互換性はないので注意してください．拡張コネクタには誤挿入防止キーが付いていて，異なる CPU や異なるグループの T-Engine 向けの拡張ボードを誤って挿入して基板を破損することがないように配慮されています．異なる T-Engine どうしで拡張ボードを共用したい場合は，PCI バスを使用すると良いでしょう．

また，PCI バスをもつ T-Engine の拡張コネクタには，ローカル・バス，PCI バスの両方が出力されているので，ローカル・バスと PCI バスの両方を同時に使用することも可能です．ただし，PCI バスが搭載された T-Engine の拡張ボードはローカル・バスの信号が一部省略されているので注意が必要です．たとえば，SH7751 の T-Engine では，データ・バスが 32 ビットのうち 16 ビット，アドレス線は 26 ビットのうち下位 18 ビットのみが出力されています．したがって，ローカル・バスの各エリアは，バス幅 16 ビットで最大 512K バイトまでしか利用できません．

● シリアル・インターフェース

T-Engine，μT-Engine では，シリアル・インターフェースの搭載を必須としています．信号レベルは RS-232-C に準拠します．汎用のシリアル・ポートとして，またはデバッグ用のコンソール・ポートとして使用することが可能です．

コネクタは，15 ピンのコネクタ（本田通信社製 RMC-EA15MY-OM15-MC1）で，表4に示すピン・アサインが規定されています．

● eTRON カード・インターフェース

カード・インターフェースは，ETSI TS102221 V4.1.0 の「VICC-Terminal Interface」に準拠した SIM カード・コネクタです．端子の配置は表5のとおりです．

● USB ホスト・インターフェース

標準 T-Engine では，USB ホスト機能を備えることが規定されています．USB Host Ver1.1 準拠（12M/1.5Mbps）の通信を行うことが可能です．

T-Kernel 上で動作するプログラムは，USB に接続したフラッシュ・メモリからブートして実行することが可能です．



T-Monitor

T-Kernel はどんな CPU にでも対応できる OS であると書きましたが，CPU に依存する部分はどこかで吸収する必要があります．それを行うのが T-Monitor と呼ばれるプログラムです．新規に T-Engine を開発する場合は，そのシステムに合う T-Monitor のみを用意すれば，T-Kernel 本体を変更する必要はありません．

T-Monitor は，電源投入後最初に起動されるプログラムで，ハードウェアの初期化や割り込みのハンドリング，そして基本的なデバッグ機能を提供します．T-Monitor のデバッグ機能を利用するには，T-Engine のシリアル・インターフェースにコンソールを接続します．

通信仕様は以下のとおりです．

- 通信速度 38,400bps または 115,200bps
- データ長 8 ビット
- ストップ・ビット 1 ビット
- パリティなし
- フロー制御 XON/XOFF
- 文字コード ASCII
- 受信行末 CR（`0x0d`）

プロローグ 1 2 3 4 5 6 7 APPENDIX1 APPENDIX2

表6 T-Monitorコマンド一覧

コマンド		コマンド説明	
Dump	D	Dump Memory	メモリ内容の表示
DumpByte	DB	Dump Memory	メモリ内容の表示
DumpHalf	DH	Dump Memory	メモリ内容の表示
DumpWord	DW	Dump Memory	メモリ内容の表示
Modify	M	Modify Memory	メモリ内容の変更
ModifyByte	MB	Modify Memory	メモリ内容の変更
ModifyHalf	MH	Modify Memory	メモリ内容の変更
ModifyWord	MW	Modify Memory	メモリ内容の変更
Fill	F	Fill Memory	メモリ内容の埋め込み
FillByte	FB	Fill Memory	メモリ内容の埋め込み
FillHalf	FH	Fill Memory	メモリ内容の埋め込み
FillWord	FW	Fill Memory	メモリ内容の埋め込み
Search	SC	Search Memory	メモリ内容のサーチ
SearchByte	SCB	Search Memory	メモリ内容のサーチ
SearchHalf	SCH	Search Memory	メモリ内容のサーチ
SearchWord	SCW	Search Memory	メモリ内容のサーチ
Compare	CMP	Compare Memory	メモリ内容の比較
Move	MOV	Move Memory	メモリ内容の転送
InputByte	IB	Input Port	I/Oポートからの入力
InputHalf	IH	Input Port	I/Oポートからの入力
InputWord	IW	Input Port	I/Oポートからの入力
OutputByte	OB	Output Port	I/Oポートへの出力
OutputHalf	OH	Output Port	I/Oポートへの出力
OutputWord	OW	Output Port	I/Oポートへの出力
Disassemble	DA	Disassemble	逆アセンブル
Register	R	Register Dump/ Modify	レジスタの表示/変更
Go	G	Go Program	プログラムの実行
BreakPoint	B	Set Break Point	ブレーク・ポイント設定
BreakClear	BC	Clear Break Point	ブレーク・ポイントのクリア
Step	S	Step Trace	ステップ・トレース実行
Next	N	Next Trace	ネクスト・トレース実行
BackTrace	BTR	Back Trace	バックトレース表示
Load	LO	Load Program/ Data	プログラム/データのロード
ReadDisk	RD	Read Disk	ディスクからの読込み
WriteDisk	WD	Write Disk	ディスクへの書込み
InfoDisk	ID	Display Disk Information	ディスク情報の表示
BootDisk	BD	Boot from Disk	ディスクからのブート
Kill	KILL	Kill Process	プロセスの強制終了
Help	H / ?	Help Message	ヘルプ・メッセージの表示
Exit	EX	Exit Monitor	モニタの終了

注: Byte: 8ビット, Half: 16ビット, Word: 32ビット.

● 送信行末CR/LF(0x0d, 0x0a)

T-Engineは，自動起動モードとモニタ起動モードの二つの起動モードを持ちます．自動起動モードでは，T-Engineは，まず起動するとハードウェアの初期化や診断を行います．その後，システムに装着されているディスク・デバイスを順に検索し，最初に見つけたブート可能なディスクからブートしてシステムを起動します．ブート可能なディスクがなかったときは，ROMに搭載されたシステムを起動します．ROMにシステムが搭載されていなかったときは，モニタのコマンド入力待ちとなります．

モニタ起動モードでは，システムを起動せずに，直接モニタのコマンド入力待ち状態に遷移します．

T-Monitorのコマンドの一覧を表6に示します．

T-Engine/μT-Engineの電源管理機能

● 電源の管理

T-Engine/μT-Engineのメインの電源の投入は，以下三つの手段で行うことができます．

1) 電源投入によるパワー・オン
2) 電源スイッチ押下によるパワー・オン
3) 拡張バスのパワー・オン信号アサートによるパワー・オン

1)の電源投入時に自動的にパワー・オンするか否かは，ディップ・スイッチなどで設定が可能です．3)の拡張バスのパワー・オン信号アサートによるパワー・オンは，たとえば拡張LANボードのWakeOnLan機能などを想定しています．

逆にパワー・オフは，電源スイッチの押下によるもののほかに，ソフトウェアによる電源の遮断が可能です．

また，T-Engine/μT-Engineには，以下の停電感知機能を実装することが規定されています．

1) 常時電源が投入されているRTCで検知する
2) バックアップ電源(UPS，バッテリなど)を利用する

バックアップ電源による停電の検知は，以下の3種類のうち，いずれかの方法で実装します．

2-1) CPUボードに停電通知信号入力端子を設ける
2-2) バッテリ電圧降下を検出する機能を設ける
2-3) 拡張ボード上に停電信号入力端子を儲け，CPUボードに通知する

● 電力モード

T-Engine/μT-Engineに搭載されているマイコンはいずれも組み込み向けに設計されており，PCに使用されているCPUのように消費電力は大きくありません．しかし，組み込み機器はバッテリで動作する場合も多く，待機時には省電力モードに移行して消費電力を抑えるなどの電源管理は非常に重要です．T-Engine/μT-Engineでは，電源の状態により以下の四つの状態を規定しています．

1) P0状態

メインの電源がOFFである状態．

2) P1状態

メインの電源はONであるが，CPUは低電力モード(スリープ)である状態．メモリの電源はOFFであり，復帰前後でRAMの内容は保存されない．周辺モジュールの状態は任意．割り込みで復帰することができる．

3) P2状態

メインの電源はONであるが，CPUは低電力モード(スリープ)である状態．メモリの電源はONであり，復帰前後でRAMの内容は保存される．周辺モジュールの状態は任意．割り込みで復帰することができる．

4) P3状態

メインの電源はONであり，CPUは通常モード．メモリの電源もONであり，RAMの内容は保存される．周辺モジュールの状態は任意．

P3状態ではさらに，CPUの動作クロックにより詳細に電力モードを規定することができます．たとえば，設定可能な動作クロックのうち，もっとも低い周波数で動作するモードをサブモード“1”とし，以下のように規定します．

低速：P3-1
中速：P3-2
高速：P3-3

T-Engine/μT-Engineでは，これらの電力モード間の遷移はソフトウェアで行うことができ，きめ細かな電力制御が可能です．

T-Engineによるハードウェア開発

● 拡張インターフェースの選択

T-Engine/μT-Engineは，T-Kernelをベースとしたソフトウェアの開発プラットホームであると同時に，組み込みシステムのハードウェアのプラットホームでもあります．最初からCPUボード上でOSが動く状態で提供されるので，CPUボードから設計する場合に比較して，大幅に設計工数を削減することが可能です．

ここでは，T-Engine/μT-Engineを使用したハードウェアの開発について説明します．一口にハードウェアといっても，その規模や要求性能はさまざまです．汎用ロジックをいくつか組み合わせて実現できるものもあれば，汎用のASSPであったり，特別に設計したASICを組み合わせる大規模なものかもしれません．

T-Engine/μT-Engineの拡張インターフェースの選択肢を図15に示します．標準T-Engineには，ユーザがハードウェアを拡張できるインターフェースとして，シリアル・インターフェース，USBホスト機能，PCMCIA，拡張コネクタがあります．μT-Engineには，USBホスト機能がなく，PCMCIAの代わりにコンパクト・フラッシュ・インターフェースとMMCまたはSDカード・インターフェースが搭載されています．T-Engine/μT-Engineにハードウェアを拡張する場合，これらインターフェースを用途に合わせ選択します．

シリアル・インターフェースは，RS-232-Cレベルの調歩同期式のデバイスを接続することができます．転送速度が速くないデバイスを接続する場合は，シリアル・インターフェースが

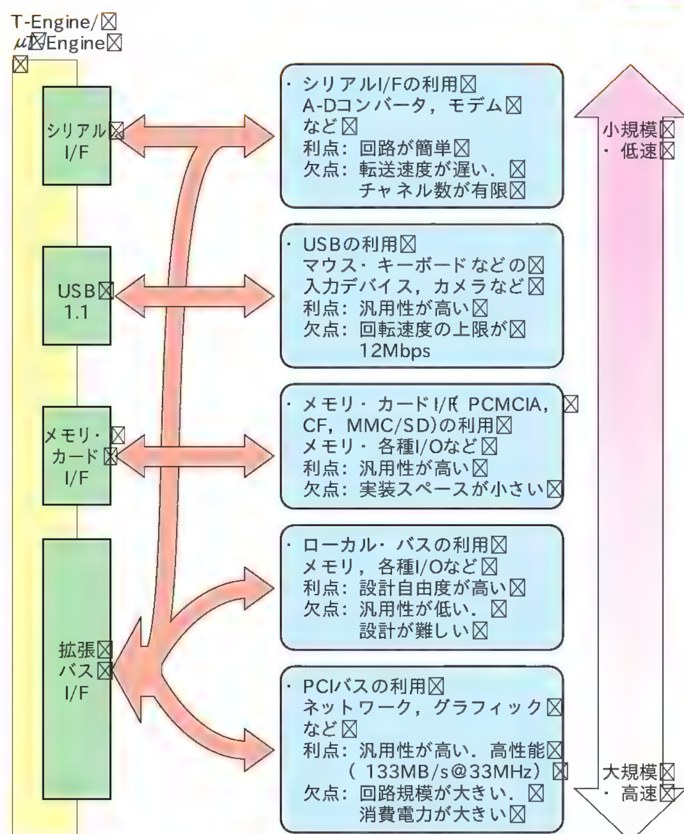


図15 T-Engineの拡張インターフェースの選択肢

便利です．ただし，T-Monitorがこのシリアル・インターフェースを用いてターミナルと接続するので排他利用になります．なお，T-Engine/μT-Engineの製品によっては拡張コネクタにシリアル・インターフェースが出力されているものもあります．

USBのインターフェースには，マウスやキーボード，カメラなどのUSBターゲット・デバイスを接続することができます．最大転送速度は12Mbpsです．また，PCMCIAやコンパクト・フラッシュなどのメモリ・カード・インターフェースを介してハードウェアを拡張することもできます．これらのインターフェースは規格化されていて，汎用性が高いというメリットがあります．

T-Engine/μT-Engineの拡張コネクタには，CPUのローカル・バスが出力されています．一部の製品にはPCIバスも合わせて出力されています．拡張コネクタを介して必要な周辺回路を実装することにより，一般的なCPU+周辺回路の構造のシステムを構成することができます．CPUのバスは伝送帯域も大きいので設計の自由度が高く，比較的大規模な回路を実装することができます．

拡張コネクタのローカル・バスに接続された拡張ボードの構造の例を図16に示します．ローカル・バスはメモリを接続することを想定しているので，拡張ボードはそのCPUによりサ

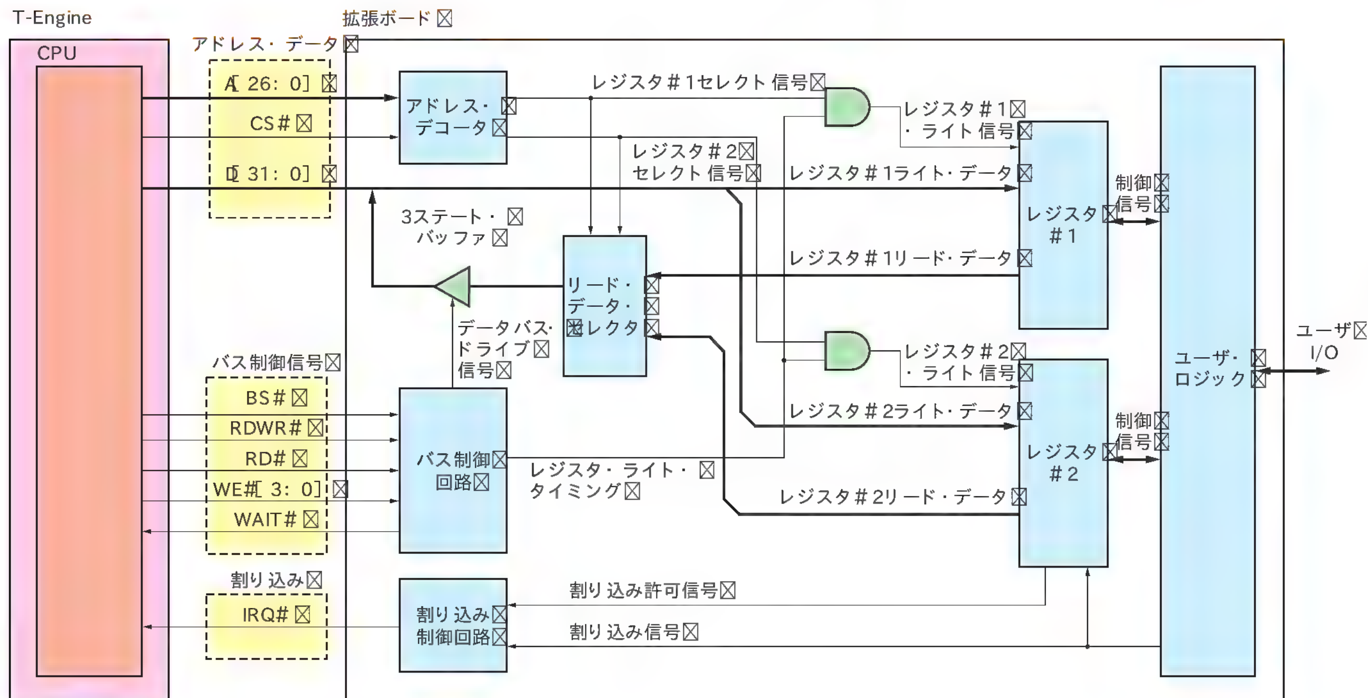


図16　ローカル・バス接続のT-Engine拡張ボード構成例

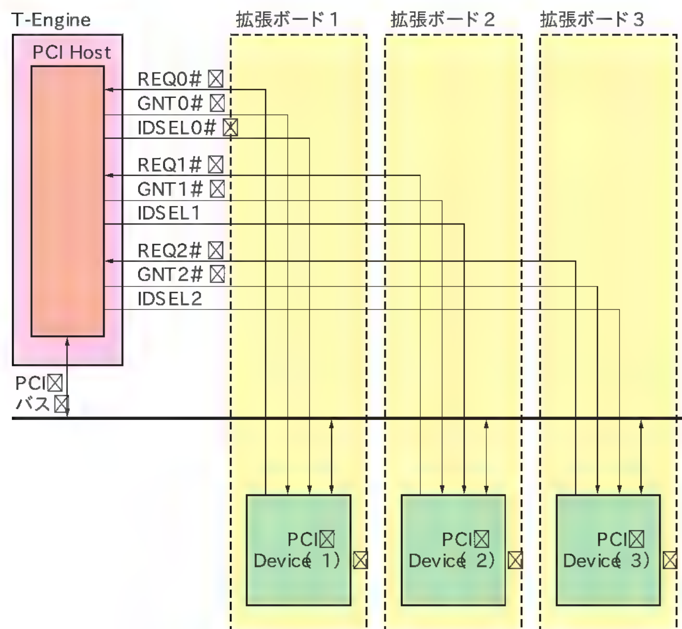


図17　T-EngineとPCIデバイスの接続例

ポートされるメモリとしてふるまう必要があります．この拡張ボードは，メモリ・マップされたレジスタをアクセスすることでユーザ・ロジックを制御します．

● PCIバスを利用したT-Engine拡張ボード

CPUのローカル・バスのプロトコルや電気的特性は，CPUのメーカや品種によって異なります．したがって，あるCPU向けに作られた拡張ボードが別のCPUに接続して動作するとは限りません．このような場合は，ローカル・バスに接続される周辺回路は接続されるCPUごとに再設計する必要があります．

PCIは，CPUのアーキテクチャに依存しない汎用のバス規格として広く使用されています．パソコンの拡張バスとしてもおなじみのPCIバスですが，最近では組み込みプロセッサにPCIバスのインターフェースを備えるものも増えてきました．標準のT-Engineでは，これまでにルネサステクノロジのSH7751R，NECエレクトロニクスのV_R5500，東芝のTX4956，μT-EngineではNECエレクトロニクスのV_R4131を使用したものが拡張コネクタにPCIが出力されています．

インターフェースにPCIを用いることの利点は，拡張ボードをより汎用にすることができることです．PCIを使用することにより，T-Engineの種類やCPUを選ばないT-Engine拡張ボードを作ることが可能です．また，PCIのインターフェースをもつ市販のLSIをより少ない外付け部品で接続することが可能になっています．ネットワーク・コントローラやグラフィック・コントローラなど，ホストとのインターフェースにPCIを持つものはたいへん多く，部品の選択肢が広がります．

また，FPGAの中には，PCIの電気的特性を満たすI/Oを持つものもあります．FPGAの大容量化にともない，PCIバス・インターフェースの論理記述がFPGAベンダから提供されていることも少なくありません．これを利用することでさらに開発期間の短縮が可能です．

基本的に，PCIのインターフェースはバス構造であり，PCIのインターフェースを備えたLSIは，PCIバスに直結できます．

ただし，PCI ホスト（セントラル・リソース）とのインターフェースはホストと PCI デバイスとの間の一対一のインターフェースです．

T-Engineと PCI デバイスとの接続例を図17に示します．この例では，PCI バス上に3個のイニシエータ・デバイスを接続した例を示しています．

PCI ホストとのインターフェースには，コンフィギュレーション・デバイス選択端子（IDSEL 端子），バス・リクエスト（REQ#），バス・グラント（GNT#）があります．IDSEL は，PCI のコンフィギュレーション・サイクルでのデバイス・セレクトに使用します．T-Engineは，IDSEL 出力を3系統持ちます．これは，イニシエータ・デバイス，ターゲット・デバイスを含め PCI デバイスを最大3個まで接続できることを意味します．逆に T-Engineの拡張バスには IDSEL 入力がないので T-Engine上の PCI 搭載マイコンは必ずホストとして動作します．

また，REQ# と GNT# は，PCI バス上のイニシエータ（バス・マスタ）の調停に使用します．PCI バス上のイニシエータは，REQ# 端子をアサートし，バス・サイクルの取得をリクエストします．PCI のホストは，REQ# に対し GNT# をアサートすることでバスの使用を許可します．T-Engineの拡張バスには，REQ#/GNT# 端子を3対持つので，最大3個のイニシエータ・デバイスを接続可能です．



T-Engine アーキテクチャの応用例

T-Engineアーキテクチャを用いた応用例として，ユビキタスネットワーキング研究所により開発されたユビキタスコミュニケータを紹介します（写真5）．

● ユビキタスコミュニケータとは？

ユビキタス社会とは，あらゆるモノや環境に配置されたコンピュータが互いに連携し，コンピュータに蓄積された知識や接続されたセンサにより得られた情報を，人々が意識することなく活用できる社会です．ユビキタス社会では，あらゆる人がストレスなくコンピュータから情報を享受することができなければなりません．ユビキタスコミュニケータは，ユビキタス社会において，人の世界とコンピュータの世界の橋渡しをするデバイスとして開発されました．

コンピュータから情報を得る手段には，状況や用途によって無線通信や赤外線通信など，さまざまなケースが考えられます．ユビキタスコミュニケータは，ユビキタス環境において「人と人」とのコミュニケーションをはじめとして「人とモノ」とのコミュニケーション，「人と環境」とのコミュニケーションを実現するための通信機能とインターフェースを備えています．

ユビキタスコミュニケータのおもな仕様を表7に示します．



写真5　ユビキタスコミュニケータ

表7
ユビキタスコミュニケータのおもな仕様

基本スペック	CPU	SH7727 96MHz
	フラッシュ・メモリ	8M バイト
	RAM	32M バイト
	外形寸法	75×120×17.6［mm］
	重量	176g（バッテリ含む）
外部インターフェース	RFID インターフェース	2.45GHz/13.56MHz マルチバンド
	USB	Host および Function
	赤外線インターフェース	送受信
	メモリカード・インターフェース	SD カード I/F 2スロット
ユーザ・インターフェース	液晶	640×480，65000色，タッチ・パネル付き
	音声	16ビット 44.1kHz，ステレオ マイク・スピーカ搭載 外部マイク入力・音声出力可能
	動画・静止画	MPEG-4，JPEG デコード・アクセラレータ
	カメラ	CIF サイズ CMOS イメージ・センサ
セキュリティ機能	生体認証	スイープ型指紋センサ
	eTRON I/F	eTRON/16対応 SIM ソケット
OS	T-Kernel/T-Kernel Extension	
ミドルウェア形式	T-Engine 規格 T-Format による流通互換形式	

1) 人とモノとのコミュニケーション

人とモノとのコミュニケーションを実現する手段として，ユビキタスコミュニケータはRFIDのインターフェースを備えています．13.56MHzと2.54GHzのマルチバンド・アンテナを搭載しており，日立製作所のミューチップや，ISO14443規格に準拠したRFIDの読み書きが可能です．

また，eTRONによる暗号通信とRFIDのインターフェースを組み合わせて電子チケットや電子マネーとして使用できます．

2) 人と環境とのコミュニケーション

降り注ぐ情報の中から周囲の状況を認識し，その人にとってそのときもっとも必要な情報を提供する，これは状況認識(context awareness)と呼ばれます．

周囲の環境を取得・制御する手段として，ユビキタスコミュニケータは赤外線通信機能を備えています．環境に配置された赤外線の送信機からの場所情報を受信し，その場所に応じた機能をもつことができます．たとえば，テレビの前に立てばテレビのリモコンとして，エアコンの前に立てばエアコンのリモコンとして動作するインテリジェント・リモコンが実現可能です．

3) 人と人とのコミュニケーション

ユビキタスコミュニケータは，MPEG-4とJPEGのアクセラレータを備えています．これにより，人間に対してより質の高い情報を提供可能です．これに携帯電話を組み合わせることによりテレビ電話を実現したり，チューナを組み合わせることによりディジタル放送の受信機を実現することも考えられます．

● ユビキタスコミュニケータとセキュリティ

ユビキタスコミュニケータは，環境や設備機器を制御できる端末であり，むやみに他人に使われると危険です．また，電子チケットや電子マネーとして使用できるので，紛失や盗難の際に悪用できないようにする必要があります．そのため，スイープ型指紋センサによる生体認証により，持ち主以外が使用できないようにしています．

このように，ユビキタスコミュニケータは，質の高いユーザ・インターフェースを実現する高い処理性能と，環境に配置されたコンピュータとのシームレスな通信を実現する多彩なインターフェースを備えています．ハードウェア設計のしやすさとT-Kernelベースのソフトウェアのリアルタイム性が遺憾なく発揮されたアプリケーションであり，T-Engineアーキテクチャを用いる利点といえるでしょう．

まとめ

T-Engineアーキテクチャが規定されるに至った背景と，そのハードウェアの概要についてまとめてきました．これまで難しかった組み込み機器の開発プラットホームの規格化が，T-Engineというハードウェア，T-Kernelという OSの強い標準化により可能になりました．

規格化団体であるT-Engineフォーラムが2002年6月に設立されてからわずか2年の間に，今や300社を超える企業・団体が加入し，活発に製品開発が行われています．T-Engine/μT-Engine開発プラットホームも続々と製品化され，国内外の主要なCPUアーキテクチャの開発プラットホームはすでに製品化されたといって差し支えないでしょう．

T-Engine/μT-Engineは"動くOS"が搭載された開発環境で，CPUまわりのトラブルで悩むことはもはやありません．

ぜひ開発に利用してみてはいかがでしょうか．

はやかわ・みき/こばやし・しんすけ/かとう・あつし
YRPユビキタスネットワーキング研究所

Information ――ターボリナックスと日本ユニシス，オープンソース・ソフトを活用した中小規模システム向けサーバの提供で協業
ターボリナックス(株)と日本ユニシス(株)は，オープンソース・ソフトウェアを活用した中小規模システム向けのサーバの提供で協業した．日本ユニシスが，ターボリナックスが開発中のサーバOS(Celica)をPCサーバに組み込んだパッケージとして販売する．

ITRONから発展し，大規模アプリケーションの開発も可能になった

第3章

リアルタイムOS T-Kernelの詳細

豊山 祐一

小規模組み込みシステムにおいてμITRONは広く普及している．組み込みシステムの実情に合わせた設計と，自由に使える仕様，そして「弱い標準化」によりシステムに最適な実装を行えることなどが高く評価されてきた．

しかし，この「弱い標準化」によりミドルウェアの流通が阻害されているとの指摘や，大規模システムのためにメモリ保護機能やプログラムのダイナミック・ロード機能が欲しいという要望が出てきた．そこで生まれたのがT-Kernelである．T-KernelはμITRONをベースとしつつも，古い仕様に縛られず新規に設計されているため，わかりやすさとあつかいやすさを両立しているという特徴がある．

本章では新たなリアルタイムOS，T-Kernelについて解説する． （編集部）

はじめに

T-KernelはT-Engineプロジェクトから生まれた新しいリアルタイムOSです．T-KernelはTRONプロジェクトにおいて長年にわたり研究・改良を続けてきたμITRONの技術をベースに，新世代のリアルタイムOSとして設計開発されました．今年の1月にT-Kernelのソース・コードは一般に向けて公開され，所定の手続きを踏めばだれでも使用することができます．

T-Kernelは，表1に示すように組み込みシステムにおいて代表的な各種CPUに対応しています．また今後もどんどん対応CPUが増えていく予定です．前章で紹介されたように，現在さまざまなCPUを使った標準T-EngineやμT-Engineなどのハードウェアが登場していますが，これらはすべてT-KernelをOSとして動いています．

また，オープン化されたT-Kernelのソース・コードを使った各種の応用製品が登場してきています（写真1）．

本章では，このT-Kernelについて詳しく説明します．



写真1　T-Engineの応用製品



T-Kernelの特徴

まずT-Kernelの特徴をμITRONとの対比を中心に説明します．

● T-KernelとμITRON

μITRONは，TRONプロジェクトの中で組み込みシステム向けOSとして開発され，現在，携帯電話，デジカメなどの最新の情報機器から各種家電製品や制御機器など，組み込みシステムの広い分野において使われています．T-Kernelは，このμITRONの優れた面を継承しつつ，急速に高度化していく組み込みシステムのニーズを満たすようにさらなる改良，拡張を行って開発されてきました．

ITRONが誕生した1980年代から，組み込みシステムの世界は大きく変わってきました．組み込みシステム向けのOSに要求されることも昔と同じではありません．T-Kernelはそれらを踏まえて，新世代の組み込みシステム向けOSをめざしています．

そのため，T-Kernelの仕様はμITRONの仕様とは異なる部分があります．これは，T-Kernelの仕様を定めるにあたって，

表1　T-Kernel（1.00.00版）対応T-EngineおよびCPU

T-Engine種別	CPU
標準T-Engine/SH7727	日立製作所/ルネサステクノロジ SH7727（SH3-DSP）
標準T-Engine/SH7751R	日立製作所/ルネサステクノロジ SH7751R（SH-4）
標準T-Engine/V_R5500	NECエレクトロニクス V_R5500（MIPS IV）
標準T-Engine/ARM920-MX1	Motorola MC9328MX1（DragonBall i.MX1, ARM920Tコア）
標準T-Engine/ARM720-S1C	EPSON S1C38000（ARM720Tコア）
μT-Engine/M32104	三菱電機/ルネサステクノロジ M32104
μT-Engine/V_R4131	NECエレクトロニクス V_R4131（MIPS II）

「μITRONの優れた点は引き継ぐが，古い仕様に縛られることなく新しいOSを作ろう」と意図されてきたからです（コラム1）．とはいっても，T-KernelはμITRONから多くを引き継いでいるので，今までμITRONを使ってきた技術者にとってはなじみやすいOSだと思われます．

● ミドルウェア流通のための強い標準化

μITRONはその基本コンセプトに「弱い標準化」があります．標準的な仕様は定めてオープンにするが，その実装や利用方法についてはユーザ側に多くが任されるという方針です．かつて組み込みシステムのCPUは，機能やパワーが貧弱であったため，実装において自由度が高いということは大きなメリットでした．逆に，プログラムの規模が比較的小さかったので，弱い標準化によるデメリットがあまり問題とされませんでした．つまり，アプリケーションの移植性よりも，多様なハードウェアやアプリケーションに対する適応化が重視されたのです．

「弱い標準化」のデメリットとは，同じμITRON仕様のOSであるにもかかわらず，メーカやCPUが異なるとOSの細かい仕様が異なってしまい，ソフトウェアの可搬性が損なわれるという点です．このため，ミドルウェアなどソフトウェア部品の標準化が進まず，μITRONにはミドルウェアが少ないとまでいわれるようになってきました．組み込みシステムのソフトウェアが今日のように複雑化，巨大化してくると，このデメリットが無視できなくなってきます．

COLUMN 1

μITRONの動向

本稿中では，従来のμITRONの問題点とそれに対するT-Kernelの対応を説明しました．ところでμITRONの側でも同様の問題点に対する取り組みが行われています．

「弱い標準化」の弊害で，μITRON仕様のOS間でアプリケーションの移植性が損なわれる問題については，μITRONではプロファイル規定で対応しています．各プロファイル規定について最低限満たすべき機能条件が定められ，同一のプロファイル規定のOS間で互換性が保証されます．μITRON4.0仕様では，スタンダード・プロファイルのほかに自動車制御用プロファイルなどが規定されています．

MMUへの対応の一部は，μITRON4.0仕様 保護機能拡張（μITRON4.0/PX仕様）でサポートされています．ただしこの仕様は名のとおり保護機能を主眼としているため，MMUもメモリ保護のために使用され，論理アドレスと物理アドレスの一致を基本としているので，T-Kernelのメモリ管理とは少々異なるものです．

このほか，μITRONの標準仕様以外に目を向ければ，各OSメーカはそれぞれのμITRONに対し，さまざまな独自拡張を行っており，実行時のプログラムのダイナミック・ロードを可能としているシステムなども存在します．

T-Engineプロジェクトではこのような問題に対応するため，T-Engineをミドルウェアの流通プラットホームとすることをめざしています．当然，そのOSであるT-Kernelは，ミドルウェアの流通を阻害しないよう，μITRONより強い標準化を行っています．

T-Kernelでは，特にハードウェアに依存した実装が必要でない限り，仕様上の実装依存を極力排除しています．このため，単に仕様を明確にするに留まらず，T-Kernelのソース・コード自体を一本化し，シングル・ソースを実現しています．T-Kernelのソース・コードの大半はC言語により記述され，異なったCPUでも同じソース・コードが使用されています．このようなことが可能になった背景には，組み込みシステムで使用されるCPUが高機能化し，OSのほとんどを高級言語で記述できるようになったことがあります．シングル・ソースはμITRONに対するT-Kernelの大きな特徴といえるでしょう．

● プログラムのダイナミック・ロード

μITRONではソフトウェアを作成する際に，OSもユーザの作成したアプリケーション・プログラムも，またミドルウェアなどのソフトウェア部品などもすべてリンクし，一つのオブジェクト・コードとするのが一般的です．その際，プログラム・コードのメモリ上の配置はスタティックに決定されます．

この手法は，パソコンのOSなどとは大きく異なります．パソコンのOSでは，WindowsでもLinuxでも，OSとアプリケーション・プログラムが一体となることはありません．アプリケーションはOSの動作中にダイナミックにロードされ，メモリ上に配置されます．

多くの組み込みシステムでは，プログラム・コードは製品が作られる段階でROM上に書き込まれ，その上で動作させていました．μITRONのように一つのオブジェクト・コードを生成する方法はこれに適しているといえます．

しかし，組み込みシステムでもPDAなど高機能な情報機器では，パソコン同様のダイナミックなロード機能が要求されるようになってきました．また，実際に実行時にダイナミック・ロードを必要とすることはなくても，ミドルウェアなどのソフトウェア部品を独立したオブジェクトとして，ロードするだけでそれが使用できることは非常に有益です．

T-Kernelではプログラムをダイナミック・ロードすることが可能です．T-Kernelではタスクや通信・同期オブジェクトなどのID番号を自動的に割り当てたり，またメモリ割り当ても動作時にダイナミックに行うなど，μITRONに比べて各種の拡張がなされています．

● MMUを用いたメモリ管理

近年，組み込みシステムでも32ビットRISCなどの高機能なCPUが使われるようになってきました．これらのCPUの多くはMMU（メモリ管理ユニット）を持っています．しかし，μITRONでは物理アドレスにおける動作を前提としているため，通常はMMUをサポートしておらず，これらの機能を使う

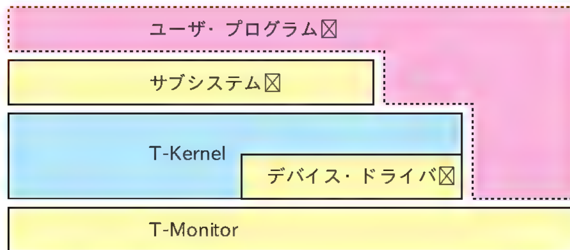


図1　T-Kernelのソフトウェア構成

ことはできませんでした．

もっとも，μITRONが物理アドレスでの動作を前提にしていることは必ずしも欠点ではありません．組み込みシステムのプログラムにおいて重要な仕事はハードウェアの制御です．ハードウェアを制御するには，物理アドレスでダイレクトにアクセスできるほうが便利です．しかも従来の組み込みシステムの小さなメモリ空間では，MMUによるメモリ管理の必要性は少なかったといえます．そもそも32ビットRISCなどが組み込みシステムで当たり前のように使われるようになったのは比較的最近のことです．

しかし，複雑化，巨大化している組み込みシステムのソフトウェアでは，MMUによるメモリ管理も求められるようになりました．前述したダイナミック・ロードにしても，ロード・オブジェクトの単位で論理アドレス空間を割り当てられれば実現が簡単になります．

またMMUのメモリ管理にはメモリの保護という側面もあります．ソフトウェア中のあるモジュールがほかのモジュールのメモリを破壊してしまうというバグは，ソフトウェアが複雑になるほど問題となります．しかし，これは各モジュールを異なった論理アドレス空間で動作させることで防ぐことができます．

T-Kernelではこれらを踏まえ，各タスクが異なった論理アドレス空間で動くことが可能となっています．

もちろんMMUのないCPUでもT-Kernelを動かすことはできますし，またシステムに応じてMMUを使用せずに物理アドレス空間で動かすことも可能です．

● より高度なシステムへの拡張性

ファイル・システムやネットワーク機能など，従来の組み込みシステムではオプション的な扱いであった機能も一般的に求められるようになってきています．μITRONではミドルウェア製品として扱われていたこれらの機能も，OSとして標準化が望まれるようになってきたのです．

しかし，すべての機能をT-Kernelに取り込むことは，OSの巨大化を招き，μITRONの長所であった「組み込みシステム向けの小さく軽量なOS」という特徴が損なわれます．組み込みシステムの世界では，ディスクレスのシステムもまだ少なくはありません．そこで，T-Kernelでは，T-Kernel Extensionという OSの機能を拡張するしくみを設けました．

T-Kernel Extensionは，T-Kernelの機能を拡張し，より高度なシステムの機能を実現するソフトウェア群です．T-Kernel ExtensionはT-Kernel上でカーネルの機能を利用して動作します．別の言い方をすれば，T-Kernelをマイクロカーネルとして，より高度なOSが実現されるわけです．また，T-Kernel Extensionにはその機能に応じていくつかの種類が存在します．ユーザが必要なExtensionを選んで使用することができるということも大きな特徴です(コラム2)．

COLUMN 2

T-Kernel Extension

T-Kernel Extensionは，Native ExtensionとPorted Extensionの二つに分類されます．

Native Extensionは，T-Engineフォーラムにより定められたT-Kernelの標準的なExtensionであり，Standard Extension，Tiny Extension，Enterprise Extensionなどが計画されています．

現在，T-Engineフォーラムで策定が進められているStandard Extensionは，独立したメモリ空間で動作するプロセスをプログラムの管理単位とし，ファイル・システムなどの上位の機能が盛り込まれたExtensionです．Standard Extension上でのプログラミングは，μITRONのプログラミングより，むしろLinuxなどのプログラミングに近いものになるかもしれません．

一方，Ported Extensionは各ベンダが開発を進めているExtensionです．現在，T-Linux，T-Java，T-Wirelessなどの名前が挙げられています．

また，同一のT-Kernel上で複数のExtensionを協調させて動かすしくみも検討されています．

T-Kernelの機能と構成

つづいてT-Kernelについてより詳細に説明していきます．

● システム・ソフトウェアの構成

T-Kernelを説明するにあたって，まずはT-KernelをOSとして用いたシステムのソフトウェア構成から説明します．T-KernelをOSとして用いたシステムとは，まさにT-Engineであり，T-Engineのソフトウェア構成がこの代表的な例となります．

図1にT-Engineのソフトウェア構成を示します．図1のとおり，T-Engineのソフトウェアは T-Kernelを含めた複数のソフトウェアから構成されます．まずT-Kernel以外のソフトウェアについて説明します．

● T-Monitor

T-Monitorは，ハードウェアの初期化やシステムの起動，割り込みのハンドリング，そして基本的なデバッグ機能など，ご

くハードウェアに近い機能を実現します．よくT-Monitorは，その機能からパソコンのBIOSに例えられます．

T-EngineのROM上には必ずT-Monitorが存在します．ハードウェアの電源が入ると，まずこのROM上のT-Monitorが起動します．T-Monitorは起動すると必要なハードウェアの初期化を行い，続いてT-Kernelの起動処理を開始します．

T-Kernelが起動した後は，T-Monitorは通常，割り込みのハンドリングや一部の機能をT-Kernelから利用されるのみで，特に意識されることはありません．

● デバイス・ドライバ

ハードウェアの制御は，原則としてデバイス・ドライバを介して行われます．デバイス・ドライバはT-Kernelの管理下に置かれ，アプリケーションはT-Kernelのサービス・コールを用いてデバイス・ドライバを呼び出します．特にアプリケーションが論理アドレス空間で動作している場合は，物理アドレス上のハードウェアはアクセスできないので，必ずデバイス・ドライバを介する必要があります．

またT-Kernelでは，ミドルウェアを流通させるためにも，デバイス・ドライバのインターフェースなどが定められています．特にT-Engineで標準的なデバイスについては標準デバイス・ドライバの仕様がT-Engineフォーラムで決められます．

● サブシステム

サブシステムはT-Kernelの機能を拡張するためのしくみです．サブシステムの実体は，サービス・コールのハンドラ（拡張SVCハンドラ）とその実行プログラムです．サブシステムをT-Kernelに登録していくことにより，T-Kernelの機能（サービス・コール）を拡張していくことができます．

前述のT-Kernel Extensionも通常はこのサブシステムの集まりとして実装されます．サブシステムにはExtensionのような上位のシステムで利用されることを前提にリソース管理の枠組みなどが実装されています．

以上のT-Monitor，各デバイス・ドライバ，各サブシステムと，T-Kernelが組み合わせられてT-Engineのシステム・ソフトウェアは実現されます．この構成は，T-Engine以外でも，T-Engineの応用製品であるT-EngineアプライアンスなどのT-Kernelを用いたシステムでも共通となります．

ユーザ・プログラムは，このシステム・ソフトウェア上で動作し，T-Monitor，T-Kernel，そしてサブシステムが提供する各機能を利用することができます．ただし，後述の保護レベルにより，ユーザ・プログラムの権限に制約を設けることも可能です．

T-Monitorやデバイス・ドライバはハードウェアへの依存性が高いため，製品のハードウェア仕様に応じた開発やカスタマイズが必要とされます．逆に，T-Monitorやデバイス・ドライバがハードウェアへの依存性を吸収してくれるおかげで，T-Kernelやその上位で動作するサブシステムやアプリケーション・ソフトウェアからはハードウェア依存性を極力排除することが可能となります．

T-Kernelの構造

つづいて，T-Kernel自身の構造について説明します．図2にT-Kernelの構造を示します．ご覧のとおりT-Kernelは，T-Kernel/OS，T-Kernel/SM，T-Kernel/DSの三つに分かれています．表2にT-Kernel/OS，SM，DSのそれぞれの機能を示します．

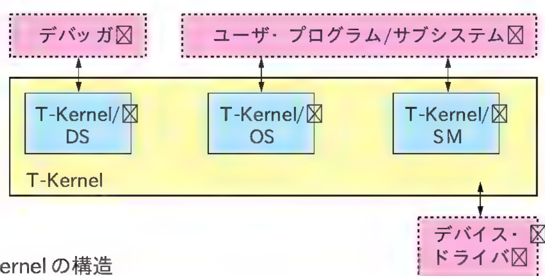


図2　T-Kernelの構造

表2　T-Kernel/OS，SM，DSの機能一覧

T-Kernel/OS	タスク管理機能 同期・通信機能 メモリ管理機能 例外/割り込み制御機能 時間管理機能 サブシステム管理機能
T-Kernel/SM	システム・メモリ管理機能 アドレス空間管理機能 デバイス管理機能 割り込み管理機能 I/Oポート・アクセス・サポート機能 省電力機能 システム構成情報管理機能
T-Kernel/DS	カーネル内部状態参照機能 実行トレース機能

● T-Kernel/OS

T-Kernelの機能の中心となるのが，このT-Kernel/OSです．OSはもちろんOperating Systemの略です．T-Kernel/OSが提供する機能がほぼμITRONの機能に相当すると考えるとわかりやすいと思います．これらの機能は原則としてサービス・コールとして提供されます．

● T-Kernel/SM

T-Kernel/SMが提供するのは，おもにμITRONにはなかったT-Kernelで拡張された機能です．SMはSystem Managerの略です．T-Kernel/SMの機能は，メモリ管理関係とハードウェア管理関係に大別できます．メモリ管理は，T-Kernel/OSではμITRONと同様のメモリ・プールの管理を行いましたが，T-Kernel/SMではMMUの仕様を前提としたメモリ空間の管理を行います．メモリ管理については次項でもう少し詳しく説明します．T-Kernel/SMの機能は，サービス・コールとライブラリ・コールの形で提供されます．ライブラリ・コールのほとんどは最終的にはT-Kernel/SMを呼び出します．

● T-Kernel/DS

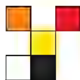
T-Kernel/DSが提供するのはデバッガなどの開発ツールのための機能です．DSはDebugger Supportの略です．T-Kernel/DSの機能を使うとT-Kernelやカーネル・オブジェクトの内部情報を参照することができます．ただし，T-Kernel/DSはデバッガなどの開発ツールから使用されることを前提としており，通常のアプリケーションからの使用は許されません．T-Kernel/DSの機能は，サービス・コールとして提供されますが，サービス・コール名も“td_”を接頭語として，ほかのコールから区別されます．通常のT-Kernelのアプリケーションのプログラミングでは T-Kernel/DSを意識する必要はありません．また，T-Kernel/DSの機能を前提にアプリケーションを作成してはいけません．

● μITRONとの機能比較

T-Kernelの機能はμITRONと比較するとわかりやすいと思います．そこで，μITRON4.0とT-Kernelの機能比較を表3にまとめてみました．μITRONの機能のほとんどにT-Kernelの機能が対応しているのがわかると思います．

μITRON4.0にあってT-Kernelにない機能として，データ・キューとオーバラン・ハンドラがあります．データ・キューについては，機能的にはメール・ボックスやメッセージ・バッファで代替が可能です．オーバラン・ハンドラについては，同等ではないものの類似の機能としてタイム・スライス時間の設定機能があります．これはタスクが一定時間動作すると待ち状態となり，同一の優先度のタスクどうしにおいてラウンドロビン・スケジューリングを実現する機能です．

表4にT-Kernelのコールの一覧を示します．表4 a)中のそれぞれのT-Kernel/OSのサービス・コールに，対応するμITRON4.0仕様のサービス・コールを記しました．多くのサービス・コールが一対一に対応していることがわかります．また，対応する名称はほとんどが，μITRONのサービス・コール名に“tk_”という接頭語をつけたものがT-Kernelのサービス・コール名となっています．この点からもμITRONの経験者はT-Kernelを理解しやすいのではと思います．ただし，始めに述べたようにT-Kernelの仕様はμITRONと完全に互換ではありません．同じ機能のサービス・コールでも詳細な仕様については確認する必要があります．



T-Kernelのメモリ管理

メモリ管理はT-Kernelの大きな特徴なので，もう少し詳しく説明します．

● MMUを前提としたメモリ管理

T-Kernelのメモリ管理はMMUの使用を前提としています．これはμITRONと比べて大きく違うところです．ただし，MMUの存在しないシステムでも，またはMMUを使用しなくてもT-Kernelは動作することが可能なので，T-KernelにはMMUが必須というわけではありません．T-Engineの仕様上，標準T-EngineではMMUは必須とされていますが，μT-Engineではオプションです．ただし，MMUを使用しない場合は，T-Kernelのメモリ管理の機能が一部無効になるという制約が生じます．

以下，MMUを使用した場合のT-Kernelのメモリ管理の機能について説明していきます．

● タスク固有空間と共有空間

T-Kernel上のソフトウェアは原則として論理アドレス空間で動作することになります．論理アドレス空間は，タスク固有空間と共有空間に分けられます．

タスク固有空間は，そこに属しているタスクのみがアクセスできるメモリ空間です．タスク固有空間はタスク生成時に指定します．同じタスク固有空間が指定されたタスクどうしはタスク固有空間を共有します．つまり，一つのタスク固有空間に複数のタスクが存在することになります．逆に一つのタスクが複数のタスク固有空間に属することはできません．

タスク固有空間はタスクのディスパッチ時に，そのタスクの属する固有空間に切り替わります．自身のタスク固有空間以外のタスク固有空間は，そのタスクの論理アドレス空間上に存在しないのでアクセスすることはできません．

共有空間は，すべてのタスクからアクセス可能なメモリ空間

表3　T-KernelとμITRON4.0の機能比較

機　能	μITRON4.0	T-Kernel
タスク管理機能	○	○
タスク付属同期機能	○	○
タスク例外処理機能	○	○
同期・通信機能	○	○
セマフォ	○	○
イベント・フラグ	○	○
データ・キュー	○	×
メール・ボックス	○	○
拡張同期・通信機能	○	○
ミューテックス	○	○
メッセージ・バッファ	○	○
ランデブ	○	○
メモリ・プール機能	○	○
固定長メモリ・プール	○	○
可変長メモリ・プール	○	○
時間管理機能	○	○
システム時刻管理	○	○
周期ハンドラ	○	○
アラーム・ハンドラ	○	○
オーバラン・ハンドラ	○	×
システム状態管理機能	○	○
割り込み管理機能	○	○
サービス・コール管理機能	○	×
システム構成管理機能	○	○
サブシステム管理機能	×	○
システム・メモリ管理機能	×	○
アドレス空間管理機能	×	○
デバイス管理機能	×	○
I/Oポート・アクセス機能	×	○
省電力機能	×	○

表4 T-Kernel コール一覧

コール名	機能	対応するμITRON4.0のコール
タスク管理機能		
tk_cre_tsk	タスクの生成	acre_tsk
tk_del_tsk	タスクの削除	del_tsk
tk_sta_tsk	タスクの起動	sta_tsk
tk_ext_tsk	自タスクの終了	ext_tsk
tk_exd_tsk	自タスクの終了と削除	exd_tsk
tk_ter_tsk	他タスクの強制終了	ter_tsk
tk_chg_pri	タスク優先度の変更	chg_tsk
tk_chg_slt	タスク・スライス・タイムの変更	—
tk_get_tsp	タスク固有空間の参照	—
tk_set_tsp	タスク固有空間の設定	—
tk_get_rid	タスクの所属リソース・グループの参照	—
tk_set_rid	タスクの所属リソース・グループの設定	—
tk_set_reg	タスク・レジスタの設定	—
tk_get_reg	タスク・レジスタの取得	—
tk_set_cpr	コプロセッサのレジスタの設定	—
tk_get_cpr	コプロセッサのレジスタの取得	—
tk_inf_tsk	タスク統計情報の参照	—
tk_ref_tsk	タスクの状態参照	ref_tsk
タスク付属同期機能		
tk_slp_tsk	自タスクを起床し待ち状態へ移行	tslp_tsk
tk_wup_tsk	他タスクの起床	wup_tsk
tk_can_wup	タスクの起床要求を無効化	can_wup
tk_rel_wai	他タスクの待ち状態の解除	rel_wai
tk_sus_tsk	他タスクを強制待ち状態へ移行	sus_tsk
tk_rsm_tsk	強制待ち状態のタスクを再開	rsm_tsk
tk_frsm_tsk	強制待ち状態のタスクを強制再開	frsm_tsk
tk_dly_tsk	タスクの遅延	dly_tsk
tk_sig_tev	タスク・イベントの送信	—
tk_wai_tev	タスク・イベントの待ち	—
tk_dis_wai	タスク待ち状態の禁止	—
tk_ena_wai	タスク待ち禁止の解除	—
タスク例外機能		
tk_def_tex	タスク例外ハンドラの定義	def_tex
tk_ras_tex	タスク例外を発生	ras_tex
tk_dis_tex	タスク例外の禁止	dis_tex
tk_ena_tex	タスク例外の許可	ena_tex
tk_end_tex	タスク例外ハンドラの終了	—
tk_ref_tex	タスク例外の状態参照	ref_tex
同期・通信機能		
セマフォ		
tk_cre_sem	セマフォの生成	acre_sem
tk_del_sem	セマフォの削除	del_sem
tk_sig_sem	セマフォ資源の返却	sig_sem
tk_wai_sem	セマフォ資源の獲得	twai_sem
tk_ref_sem	セマフォの状態参照	ref_sem
イベント・フラグ		
tk_cre_flg	イベント・フラグの生成	acre_flg
tk_del_flg	イベント・フラグの削除	del_flg
tk_set_flg	イベント・フラグのセット	set_flg
tk_clr_flg	イベント・フラグのクリア	clr_flg
tk_wai_flg	イベント・フラグの待ち	twai_flg
メール・ボックス		
tk_cre_mbx	メール・ボックスの生成	acre_mbx
tk_del_mbx	メール・ボックスの削除	del_mbx
tk_snd_mbx	メール・ボックスへ送信	snd_mbx
tk_rcv_mbx	メール・ボックスから受信	trcv_mbx
tk_ref_mbx	メール・ボックスの状態参照	ref_mbx
拡張同期・通信機能		
ミューテックス		
tk_cre_mtx	ミューテックスの生成	acre_mtx
tk_del_mtx	ミューテックスの削除	del_mtx
tk_loc_mtx	ミューテックスのロック	tloc_mtx
tk_unl_mtx	ミューテックスのアンロック	unl_mtx
tk_ref_mtx	ミューテックスの状態参照	ref_mtx
メッセージ・バッファ		
tk_cre_mbf	メッセージ・バッファの生成	acre_mbf
tk_del_mbf	メッセージ・バッファの削除	del_mbf
tk_snd_mbf	メッセージ・バッファへ送信	tsnd_mbf
tk_rcv_mbf	メッセージ・バッファから受信	trcv_mbf
tk_ref_mbf	メッセージ・バッファの受信状態	ref_mbf
ランデブ		
tk_cre_por	ランデブ・ポートの生成	acre_por
tk_del_por	ランデブ・ポートの削除	del_por
tk_cal_por	ランデブ・ポートに対するランデブの呼び出し	tcal_por
tk_acp_por	ランデブ・ポートに対するランデブ受け付け	tacp_por
tk_fwd_por	ランデブ・ポートに対するランデブの回送	fwd_por
tk_rpl_rdv	ランデブの返答	rpl_por
tk_ref_por	ランデブ・ポートの状態参照	ref_por
メモリ・プール管理機能		
固定長メモリ・プール		
tk_cre_mpf	固定長メモリ・プールの生成	acre_mpf
tk_del_mpf	固定長メモリ・プールの削除	del_mpf
tk_get_mpf	固定長メモリ・ブロックの獲得	tget_mpf
tk_rel_mpf	固定長メモリ・ブロックの返却	rel_mpf
tk_ref_mpf	固定長メモリ・プールの状態参照	ref_mpf
可変長メモリ・プール		
tk_cre_mpl	可変長メモリ・プールの生成	acre_mpl
tk_del_mpl	可変長メモリ・プールの削除	del_mpl
tk_get_mpl	可変長メモリ・ブロックの獲得	tget_mpl
tk_rel_mpl	可変長メモリ・ブロックの返却	rel_mpl
tk_ref_mpl	可変長メモリ・プールの状態参照	ref_mpl
時間管理機能		
システム時刻		
tk_set_tim	システム時刻の設定	set_tim
tk_get_tim	システム時刻の取得	get_tim
tk_get_otm	システム稼働時間の参照	—
周期ハンドラ		
tk_cre_cyc	周期ハンドラの生成	acre_cyc
tk_del_cyc	周期ハンドラの削除	del_cyc
tk_sta_cyc	周期ハンドラの動作開始	sta_cyc
tk_stp_cyc	周期ハンドラの動作停止	stp_cyc
tk_ref_cyc	周期ハンドラの状態参照	ref_cyc
アラーム・ハンドラ		
tk_cre_alm	アラーム・ハンドラの生成	acre_slm
tk_del_alm	アラーム・ハンドラの削除	del_alm
tk_sta_alm	アラーム・ハンドラの動作開始	sta_alm
tk_stp_alm	アラーム・ハンドラの動作停止	stp_alm
tk_ref_alm	アラーム・ハンドラの状態参照	ref_alm
システム状態管理機能		
tk_rot_rdq	タスクの優先順位の回転	rot_tsk
tk_get_tid	実行状態タスクのタスク ID 参照	get_tid
tk_dis_dsp	ディスパッチ禁止	dis_dps
tk_ena_dsp	ディスパッチの許可	ena_dsp
tk_ref_sys	システム状態の参照	ref_sys
tk_set_pow	省電力モードの設定	—
tk_ref_ver	バージョンの参照	ref_ver
割り込み管理機能		
tk_def_int	割り込みハンドラの定義	def_inh
tk_ret_int	割り込みハンドラから復帰	—
サブシステム管理機能		
tk_def_ssy	サブシステムの定義	—
tk_sta_ssy	スタート・アップ関数の呼出し	—
tk_cln_ssy	クリーン・アップ関数の呼出し	—
tk_evt_ssy	イベント処理関数の呼出し	—
tk_ref_ssy	サブシステム定義情報の参照	—
tk_cre_res	リソース・グループの生成	—
tk_del_res	リソース・グループの削除	—
tk_get_res	リソース管理ブロックの取得	—

(a) T-Kernel/OS コール一覧

Information ――シマンテック，5月のウィルス報告件数ランキングを発表，国内1位はガオボット，ワールド・ワイドではサッサー
(株)シマンテックは，同社に寄せられた5月のウィルス報告件数ランキングを発表した．国内1位は「W32.HLLW.Gaobot.Gen(ガオボット)」，2位は「W32.Sasser.C.Worm(サッサー)」だった．ワールド・ワイドでは，サッサーが1位に．

表4 T-Kernelコール一覧(つづき)

コール名	機能
システム・メモリ管理機能	
tk_get_smb	システム・メモリの割り当て
tk_rel_smb	システム・メモリの解放
tk_ref_smb	システム・メモリ情報取得
Vmalloc	非常駐メモリの割り当て
Vcalloc	非常駐メモリの割り当て
Vrealloc	非常駐メモリの再割り当て
Vfree	非常駐メモリの解放
Kmalloc	常駐メモリの割り当て
Kcalloc	常駐メモリの割り当て
Krealloc	常駐メモリの再割り当て
Kfree	常駐メモリの解放
アドレス空間管理機能	
SetTaskSpace	タスクのアドレス空間設定
ChkSpaceR	メモリ読み込みアクセス権の検査
ChkSpaceRW	メモリ読み込み書き込みアクセス権の検査
ChkSpaceRE	メモリ読み込みアクセス権および実行権の検査
ChkSpaceBstrR	文字列読み込みアクセス権の検査
ChkSpaceBstrRW	文字列読み込み書き込みアクセス権の検査
ChkSpaceTstrR	TRONコード文字列読み込みアクセス権の検査
ChkSpaceTstrRW	TRONコード文字列読み込み書き込みアクセス権の検査
LockSpace	メモリ領域のロック
UnlockSpace	メモリ領域のアンロック
CnvPhysicalAddr	物理アドレスの取得
MapMemory	メモリのマップ
UnmapMemory	メモリのアンマップ
デバイス管理機能	
tk_def_dev	デバイスの登録
tk_ref_idv	デバイス初期情報の取得
tk_opn_dev	デバイスのオープン
tk_cls_dev	デバイスのクローズ
tk_rea_dev	デバイスの読み込み開始
tk_srea_dev	デバイスの同期読み込み
tk_wri_dev	デバイスの書き込み開始
tk_swri_dev	デバイスの同期書き込み
tk_wai_dev	デバイスの要求完了待ち
tk_sus_dev	デバイスのサスペンド
tk_get_dev	デバイスのデバイス名取得
tk_ref_dev	デバイスの情報取得
tk_oref_dev	デバイスの情報取得
tk_lst_dev	登録済みデバイス一覧の取得
tk_evt_dev	デバイスにドライバ要求イベントを送信
割り込み管理機能	
DI	外部割り込み禁止
EI	外部割り込み許可
isDI	外部割り込み禁止状態の取得
DINTNO	割り込みベクタから割り込み定義番号へ変換
EnableInt	割り込み許可
DisableInt	割り込み禁止
ClearInt	割り込み発生のクリア
EndOfInt	割り込みコントローラにEOI発行
CheckInt	割り込み発生の検査
I/Oポート・アクセス・サポート機能	
out_b	I/Oポート書き込み(バイト)
out_h	I/Oポート書き込み(ハーフワード)
out_w	I/Oポート書き込み(ワード)
in_b	I/Oポート読み込み(バイト)
in_h	I/Oポート読み込み(ハーフワード)
in_w	I/Oポート読み込み(ワード)
WaitNsec	微小待ち(ns)
WaitUsec	微小待ち(μs)
省電力機能	
low_pow	システムを低消費電力モードに移行
off_pow	システムをサスペンド状態に移行
システム構成情報管理機能	
tk_get_cfn	システム構成情報から数値列取得
tk_get_cfs	システム構成情報から文字列取得

(b) T-Kernel/SMコール一覧

コール名	機能
カーネル内部状態取得機能	
td_lst_tsk	タスクIDのリスト参照
td_lst_sem	セマフォIDのリスト参照
td_lst_flg	イベント・フラグIDのリスト参照
td_lst_mbx	メール・ボックスIDのリスト参照
td_lst_mtx	ミューテックスIDのリスト参照
td_lst_mbf	メッセージ・バッファIDのリスト参照
td_lst_por	ランデブ・ポートIDのリスト参照
td_lst_mpf	固定長メモリ・プールIDのリスト参照
td_lst_mpl	可変長メモリ・プールIDのリスト参照
td_lst_cyc	周期ハンドラIDのリスト参照
td_lst_alm	アラーム・ハンドラIDのリスト参照
td_lst_ssy	サブシステムIDのリスト参照
td_rdy_que	タスクの優先順位の参照
td_sem_que	セマフォの待ち行列の参照
td_flg_que	イベント・フラグの待ち行列の参照
td_mbx_que	メール・ボックスの待ち行列の参照
td_smbf_que	メッセージ・バッファ送信待ち行列の参照
td_rmbf_que	メッセージ・バッファ受信待ち行列の参照
td_cal_que	ランデブ呼び出し待ち行列の参照
td_acp_que	ランデブ受け付け待ち行列の参照
td_mpf_que	固定長メモリ・プールの待ち行列の参照
td_mpl_que	可変長メモリ・プールの待ち行列の参照
td_ref_tsk	タスクの状態参照
td_ref_sem	セマフォの状態参照
td_ref_flg	イベント・フラグの状態参照
td_ref_mbx	メール・ボックスの状態参照
td_ref_mtx	ミューテックスの状態参照
td_ref_mbf	メッセージ・バッファの状態参照
td_ref_por	ランデブ・ポートの状態参照
td_ref_mpf	固定長メモリ・プールの状態参照
td_ref_mpl	可変長メモリ・プールの状態参照
td_ref_cyc	周期ハンドラの状態参照
td_ref_alm	アラーム・ハンドラの状態参照
td_ref_ssy	サブシステムの状態参照
td_ref_tex	タスク例外の状態参照
td_inf_tsk	タスク統計情報の参照
td_get_reg	タスク・レジスタの参照
td_set_reg	タスク・レジスタの設定
td_ref_sys	システムの状態参照
td_get_tim	システム時刻の参照
td_get_otm	システム稼働時間の参照
デバッグ用名称機能	
td_ref_dsname	デバッグ用名称の参照
td_set_dsname	デバッグ用名称の設定
実行トレース機能	
td_hoc_svc	システム・コール・拡張SVCのフック・ルーチン定義
td_hoc_dsp	タスク・ディスパッチのフック・ルーチン定義
td_hoc_int	割り込みハンドラのフック・ルーチン定義

(c) T-Kernel/DSコール一覧

です．タスク間で共有するメモリ領域は，通常は共有空間に配置します．注意すべき点は，共有空間はすべてのタスクからアクセスが可能であるという点です．特定のタスクのグループのみに共有な空間は設定できません．

共有空間上の未使用のメモリ領域は，通常T-Kernel/SMのシステム・メモリ管理機能により管理されます．共有メモリ上の領域が必要になった場合，T-Kernel/SMを介して動的に確保し，不用になれば，T-Kernel/SMを介して解放します．T-Kernelの内部で使用されているメモリや，メッセージ・バッファ，メモリ・プールのメモリ領域，タスクに自動割り当てされるスタックなどは，すべてこのように動的に割り当てられます．

T-Kernelなどシステム自体が使用しているメモリ空間は，ハードウェアや実装にも依存しますが，原則として共有空間上に存在します．

● タスクの論理アドレス空間

あるタスクのプログラムから見た論理アドレス空間は，自身のタスクが属するタスク固有空間と共有空間から成ります．複数のタスクが存在する場合，共有空間はどのタスクからも同じメモリ領域ですが，タスク固有空間は異なります．ただし，同じタスク固有空間に属するタスクはこの限りではありません．

一例を図3に示します．この例では，タスクBとタスクCは同じタスク固有空間に属し，タスクAは異なったタスク固有空間に属しています．よって，タスクAはタスク固有空間#1と共有空間にのみアクセスが可能です．タスクBとタスクCはタスク固有空間#2と共有空間にのみアクセスが可能となります．タスクに異なったタスク固有空間を割り与えることにより，あるタスクが関係のないタスクのメモリを破壊するといったことは防ぐことができます．その代わり，グローバル変数であっても，タスク固有空間が異なればアクセスすることはできなくなります．μITRONのプログラムでは，すべてのタスクが同じ物理アドレス空間で動作しているという前提でグローバル変数を使用しているものもあります．このようなプログラムは，T-Kernelに移植した際に問題となるので注意が必要です．

● タスク固有空間の管理

T-Kernelでは，共有空間はT-Kernel/SMのシステム・メモリ管理機能により管理されますが，実はタスク固有空間の管理機能はありません．T-Kernelが提供している機能は，基本的にタスク固有空間の切り替えのみです．タスク固有空間の切り替えが行われるのは，タスクのディスパッチ，またはサービス・コールにより要求された場合です．タスク固有空間は，具体的にはタスク固有空間ページ・テーブルと論理空間IDにより指定されます．空間の切り替えはこの二つのデータを基にMMUに設定が行われます．

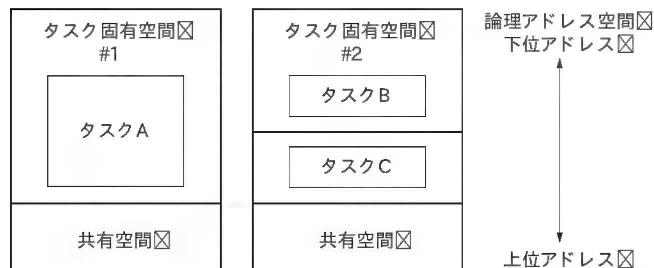


図3 タスク固有空間と共有空間

表5 tk_set_tspの仕様

tk_set_tsp タスク固有空間の設定	
書式	ER tk_set_tsp(ID tskid, T_TSKPSC *pk_tskpsc);
引き数	ID tskid 対象とするタスクのID T_TSKPSC *pk_tskpsc タスク固有空間の指定 typedef struct { VP uatb; タスク固有空間ページ・テーブルのアドレス INT lsid; タスク固有空間ID } T_TSKPSC;
返り値	エラー・コード
機能	tskidで指定されたタスクのタスク固有空間を，pk_tskspcで指定されたタスク固有空間に変更する

タスク固有空間を変更するtk_set_tspコールの仕様を表5に示します．第1引き数tskidで指定されたタスクの固有空間が，第2引き数pk_tskspcで指定されたタスク固有空間に変更されます．pk_tskspcのメンバであるuatbとlsidにより，MMUの設定がハードウェアの仕様に応じて行われます．たとえば，CPUがSH7727の場合は，uatbの値がMMUのTTBレジスタに設定され，lsidの値がMMUのPTEHレジスタに設定されます．

なお，T-Kernelはタスク固有空間の変更による影響を感知しないので，変更には注意が必要です．実行中のタスクのコードやデータがタスク固有空間にある状態で，これを変更すると，そのタスクの暴走を招きます．通常，アプリケーションのレベルのタスクは，タスク固有空間を変更する必要はありません．

実際にタスク固有空間を使用するには，この空間を生成したり割り当てたりする機能が必要となりますが，先に述べたとおり，その機能はT-Kernelには存在しません．実は，この機能はT-Kernel Extensionのほうでサポートすることになっています．たとえば，T-Kernel Standard Extensionでは，Linuxなどでいうようなプロセス管理が実現されています．つまり，T-Kernelがサポートするタスク固有空間は，上位のシステムにおいてプロセス管理のようなより高度な管理を実現するための機能と考えてもよいでしょう．

では，T-Kernel ExtensionなしにT-Kernelのみの上でプログラムを動かす場合にはどうしたらよいかというと，一般的にはタスク生成時にタスク固有空間の指定を行いません．タスク固有空間が指定されなかったタスクは，共有空間のみが使用可能となります．

● メモリの常駐/非常駐

仮想メモリは，ディスクなどの外部記憶にメモリ領域をスワップすることにより，実メモリ以上のアドレス空間を持つことができる機能です．ただし，メモリのスワップは，ディスクへのアクセスなどの処理が発生するため，動作の制約やリアルタイム性の低下が起こります．そこでT-Kernelでは，メモリ領域に常駐と非常駐の指定を行うことができます．常駐メモリに指定された領域はつねに実メモリ上に存在し，スワップすることはありません．

なお，仮想メモリ管理はT-Kernel自体の機能には含まれません．仮想メモリ管理を実現するためにはメモリ管理に直結したファイル・システムなどが必要です．この機能はT-Kernel Standard Extensionなどの上位のシステムで実装されます．つまり，T-Kernelにおけるメモリの常駐/非常駐の指定は，上位

のシステムで仮想メモリ管理を実現するための機能といえます．

● メモリの保護レベル

T-Kernelのメモリ管理機能には，メモリの保護機能があります．これは，今まで述べてきた論理アドレス空間の管理とは別物なので混同しないようにしてください．

メモリ領域には保護レベルというものが設定されます．保護レベルはレベル0からレベル3までの4段階があり，値が小さいほど上位となります．

そしてタスクにも生成時に保護レベルを設定できます．タスクの保護レベルは，むしろメモリの保護レベルに対応した実行レベルといったものです．実行レベルNのタスクは，保護レベルN以下のメモリ領域にアクセスすることができます．

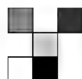
たとえば，実行レベル2のタスクは，保護レベル2と3のメモリにアクセスができます．保護レベル1に対してはアクセスできません．共有空間上のメモリ領域でも，保護レベルが異なればアクセスできません．

保護レベルは表6に示すように用途が決められています．保護レベルは，アプリケーションのプログラムがシステムのメモリ領域を破壊するなどといったことを防ぐことを想定しています．

保護レベルによるメモリの保護は，CPUやMMUなどのハードウェアの機能を使って実現されます．保護レベルに違反したアクセスが行われた場合，ハードウェアによる例外(割り込み)が発生します．よって，実際のメモリ保護の機能はハードウェアに依存します．多くのCPUでは，4レベルのメモリ保護を実現することはできません．たとえば，特権モードとユーザ・モードといった二つのレベルしかサポートされていません．

そこで，T-Kernelの四つの保護レベルは，ハードウェアの持つレベルに割り当てられることとなっています．ハードウェアの持つレベルがレベル2の場合は，レベル0からレベル2が特権モードに，レベル3がユーザ・モードに割り付けられることになります．このような場合，プログラム上は保護レベルが4段階に設定できても，動作上はレベル3以外は同じメモリ保護となります．また，MMUなどを持たないCPUではメモリ保護の機能自体がないので，すべてのレベルが同等になってしまいます．しかし，保護レベルを適切に設定することは，プログラムをほかのCPUに移植する際に重要となるので，このルールは守る必要があります．

保護レベルにはメモリ保護以外の機能もあります．T-Kernelの構築時に，ある保護レベル以下のタスクからT-Kernelのサービス・コールの発行を禁止することが可能です．この機能はT-Kernel Extensionを使用する際に，ユーザ・レベルのタスクから直接T-Kernelを利用することを禁止し，Extensionが提供する機能のみを利用可能にする際に使用されます．



T-Kernelプログラミング

T-Kernel上で動くソフトウェアのプログラミングについて説明します．

表6　保護レベルの用途

レベル	用　途
レベル0	システム・ソフトウェア(OS，デバイス・ドライバなど)
レベル1	システム・アプリケーション
レベル2	未使用(予約)
レベル3	ユーザ・アプリケーション

● T-Kernelのプログラミング

T-Kernelの機能は今まで見てきたとおり，μITRONの機能をほぼサポートしています．したがって，やろうと思えばT-KernelをμITRONとほぼ同等に使用することもできます．つまり，今までのμITRONの代わりにT-Kernelを単純に置き換え可能です．たとえば，T-Kernel上のすべてのタスクを保護レベル0とし，MMUを使用せずに物理アドレス空間で動作させれば，μITRONのプログラミングとの差はほとんどありません．このようなT-Kernelの使い方も製品開発によっては必要かもしれません．ただし，μITRON流の使い方ではT-Kernelのすべての機能を利用することはできません．可能な限り，これから述べるT-Kernelの作法に従ってプログラミングすることをお勧めします．このようにして作られたプログラムは，ハードウェアへの依存性が少なく，再利用性の高いものとなります．

● ソフトウェアの形態

まず，作成するプログラムが，アプリケーション(ユーザ・プログラム)，デバイス・ドライバ，サブシステムのどの形態をとるか決めることが重要です．実際にはある程度の規模のプログラムは，これらアプリケーション，デバイス・ドライバ，サブシステムが複数組み合わせられて一つのソフトウェアを構成することになります．

μITRONのプログラミングではアプリケーションとデバイス・ドライバの区別などが明確ではない場合もありましたが，T-Kernelでははっきり区別されます．ハードウェアを制御するソフトウェアを原則としてデバイス・ドライバとなります．

ある程度一般的な機能を提供するソフトウェアは，サブシステムとし，サービス・コールとして機能を提供できるようにします．ファイル・システムやネットワーク機能などは代表的なサブシステムです．

デバイス・ドライバはハードウェア単位のソフトウェア部品であり，サブシステムは機能単位のソフトウェア部品と考えても良いでしょう．

また，多くのサブシステムは，その中でハードウェアを使用します．そのようなときはサブシステムからデバイス・ドライバの機能を呼び出すのが本来の方法ですが，効率を優先してサブシステムから直接ハードウェアを制御することもあります．サブシステムはデバイス・ドライバ同様，システム・ソフトウェアの一部なので，ハードウェアのアクセスは可能です．ただし，サブシステムの可搬性を考えるならば，可能な限りハー

ドウェアへの制御はデバイス・ドライバを介して行うべきです．

アプリケーションは，デバイス・ドライバやサブシステムと異なり，ユーザ・レベルのプログラムです．ハードウェアへの直接のアクセスは許されません．これはアプリケーションがハードウェアにほとんど依存しないことを意味します．

● メモリ管理を意識したプログラミング

T-Kernelのプログラミングを行ううえで重要なことは，T-KernelがMMUを用いたメモリ管理を行っているということを意識することです．これから述べる点に注意してプログラミングしてください．各タスクは異なった論理アドレス空間で動いている可能性があり，タスクやメモリ領域ごとに保護レベルが設定されています．このことはμITRONのプログラミングに慣れている人が誤りやすい点です．

以下に注意すべき点を述べていきます．

- アプリケーションのタスクからハードウェアをアクセスしない

ハードウェアは，必ずデバイス・ドライバまたはサブシステムを介して制御します．

- グローバル・データを安易に使用しない

タスク間の通信はT-Kernelの同期・通信機能を利用します．また，共有したいデータは共有空間から動的に確保して使用します．

- ほかのタスクから受け取ったポインタは注意が必要

ほかのタスクは異なった論理アドレス空間で動いている可能性があります．特にデバイス・ドライバやサブシステム側のプログラムを作成する際は，アドレス変換やタスク固有空間の変更が必要です．

- サブシステムではサービス要求元のメモリのアクセス権をチェックする

サービス要求元のタスクの保護レベルが，要求したサービスによるメモリ・アクセスが許されているか否かを事前にチェックする必要があります．

- 非常駐メモリは，実メモリ上に存在しない場合がある

通常のプログラムの実行では意識する必要はありませんが，割り込みハンドラやDMAからアクセスする場合は常駐化が必要です．

もちろん，MMUを使用していないシステムの場合，以上の点を無視してもプログラムは動作します．しかし，これらを無視するということは，T-Kernelの利点であるプログラムの可搬性，再利用性を失うことになります．つまり，MMUを使用したシステム上では動作できなくなります．逆に，MMUを使用しないシステム上であっても，MMUを意識したプログラムは動作が可能です．たとえば，保護レベルの設定やチェックの機能は，MMUを使用しないシステム上で呼び出しても問題はありません．実際のメモリ保護がそのシステム上では行われないというだけです．つまり，実際にMMUが使用されているか否かに関係なくMMUを意識してプログラミングされることが望ましいわけです．

表7 T-KernelとμITRON4.0のイベント・フラグ関係コールの対比

機　能	T-Kernel	μITRON4.0
イベント・フラグの生成	tk_cre_flg	acre_flg
イベント・フラグの削除	tk_del_flg	del_flg
イベント・フラグのセット	tk_set_flg	set_flg
イベント・フラグのクリア	tk_clr_flg	clr_flg
イベント・フラグ待ち	tk_wai_flg	twai_flg
イベント・フラグの状態参照	tk_ref_flg	ref_flg

● μITRONからT-Kernelへの移植

T-Kernelを使い始めるにあたり，まず今までμITRONで作ってきたプログラムを移植してみようという方は多いと思います．そこでμITRONからT-Kernelにソフトウェアを移植する方法について説明します．

ソフトウェアの移植にはいくつかのレベルが考えられます．

1) 単純にOSのサービス・コールをμITRONからT-Kernelに変えたレベル

とりあえず動かしたというレベルです．動作条件はMMU未使用で物理アドレス空間となります．可搬性も乏しくなります．

2) 上記の1)に加えて，T-Kernelの仕様に基づき，サブシステム化やデバイス・ドライバ化を実施したレベル

動作条件はMMU未使用で物理アドレス空間ですが，その条件において可搬性は出てきます

3) 上記の1)と2)に加えて，MMU利用環境にも対応したレベル

これが望ましいT-Kernelソフトウェアです．可搬性に優れ，T-Kernelのミドルウェアとして流通が可能です．

まず1)のレベルのサービス・コールの変換は，μITRONとT-Kernelのサービス・コールの多くが一対一に対応しているので容易だと思います．

例として，表7にμITRON4.0とT-Kernelのイベント・フラグ関係のサービス・コールの対比を示します．ほとんどがそのまま置き換え可能なことがわかります．その際に一番問題となるのはカーネル・オブジェクトのID番号でしょう．表の中でも，イベント・フラグの生成コールが，μITRON側がcre_flgコールではなくacre_flgコールであることに着目してください．両者の違いは，フラグのIDを前者はユーザ(プログラマ)が指定するのに対し，後者はプログラムの実行時に自動的に割り当てられる点です．

μITRONではID番号はプログラミング時に任意の値をつけることが可能でしたが，T-Kernelではすべて動作時に動的に割り当てられます．したがって，後でIDを使用する必要がある場合は，IDを記録するようにプログラムを修正する必要があります．

2)のレベルのサブシステム化，デバイス・ドライバ化は大なり小なり元のプログラムの構造を変える必要があります．変更

量は元のプログラムの作りに左右されます．ハードウェアの制御部がデバイス・ドライバとして独立しており，また汎用的な機能が拡張SVCハンドラとして実装されているプログラムならば，インターフェースの変更程度で比較的スムーズに移植できると考えられます．

3)のレベルのMMU利用環境の対応は，プログラム全体の見直しが必要です．プログラムの変更よりも，メモリのアクセス違反がないか，適切なアドレス・チェックは行われているか，などのチェックがメインの作業となります．また，サブシステムについては必要に応じてタスク固有空間やリソース・グループの切り替えなどが必要となってきます．

どのレベルまで移植を行うかは，その移植の目的により異なります．移植の作業量と，各レベルの利点や制約を考慮して決定する必要があります．

● T-Format

T-Formatはミドルウェア流通のためにT-Engineフォーラムが規定している規則です．複数のベンダが提供するミドルウェアを組み合わせてシステムを構築できるように，ファイル形式やグローバルな名前の命名規則などを定めています．

たとえばグローバルな名称については，各ベンダごとに割りふられたベンダ・コードが頭につきます．

もし，商業ベースでT-Kernelのミドルウェアの開発を考えているのならば，T-Formatに従うことが重要です．現時点では，T-FormatはT-Engineフォーラム内部でのみ公開されているので，利用する際にはT-Engineフォーラムへの参加が必要となります．

ソース・コードの入手とオブジェクトの構築

公開されているT-Kernelのソース・コードと，それを使ってT-Kernelのオブジェクトを構築する方法を説明します．

● ソース・コードの公開について

T-Kernelのソース・コードは，T-EngineフォーラムのWebページ(http://www.t-engine.org/)より，一般に公開されています(図4)．正規の手続きを踏めばだれでも入手して使うことができます．T-Kernelは実際に組み込みシステムの製品に使用されることを前提にしているので，これを製品に組み込んで使用することも可能ですし，その際にT-Kernelを自由に改造することもできます．

この項ではT-Kernelを入手し，実際に動かしてみるまでを説明します．

● T-License

T-Kernelのソース・コードを入手するには，T-EngineフォーラムのWebページよりT-LicenseというT-Kernelのライセンス契約を締結し，ソース・コードの利用を申し込むことが必要です．T-Licenseの詳細については，このWebページで実物を



図4　T-KernelのWebページ

読んでもらうことにして，ここではT-Licenseの特徴について簡単に説明します．

一般に公開されたソース・コードのライセンスとしては，GPL(General Public License)が有名です．しかしT-Licenseは，これらいわゆるオープン・ソースのライセンスとは異なっています．GPLでは元のプログラムを改変した場合，そのソース・コードを開示する必要があります．一方，T-Licenseではこのような開示の義務はありません．逆に改変したソース・コードを勝手に配布することは禁止されています．これはT-LicenseがGPLとは別の考え方に基づくものだからです．

まず，T-Licenseは組み込みシステムの製品への使用を第一に考えています．組み込みシステムの場合，ソース・コードはそのハードウェアと密接な関係にあり，製品のノウハウと直結しています．そのため，メーカでは製品に使用しているソフトウェアのソース・コードの公開を避けたがる傾向にあります．これはGPLのソフトウェアを製品に使用する際に，しばしば問題となります．GPLなどのオープン・ソースでは大勢の人間が参加してソフトウェアを開発，改良していくことをめざしているわけですが，組み込みシステムの製品開発には必ずしも適してはいないと思われます．

誤解のないよう繰り返しますが，T-LicenseとGPLはどちらが優れているといったものではなく，目的と対象が異なるということです．T-Licenseは，T-Engineフォーラムに集まった組み込みシステムのメーカの意見を元に，組み込みシステムの製品において使用しやすいソース・コードのライセンスとなっています．

次に，T-Engineのプロジェクトは，ミドルウェアの流通を重要視しています．その際，たとえCPUを含むハードウェアが異なってもOSの互換性が保たれることが重要です．もし，いろいろなハードウェアにカスタマイズされた改変版T-Kernelが世に広まれば，μITRONと同様に同じOSなのにソフトウェ

アの移植性がないという状況が発生します．これは，比較的ハードウェアが標準化され，適用分野が限定されているパソコンの世界とは異なる点です．

以上の点を踏まえ，T-Licenseでは改変版のソース・コードの配布を禁じ，T-Kernelはオリジナルのソース・コードのみがT-Engineフォーラムより配布されるよう定めています．

誤解してはいけないのは，T-Licenseは改変自体を禁止しているのではない，ということです．T-Kernelを改変すること自体は自由です．改変したT-Kernelを製品に使用してもかまいません．禁止されているのは，改変されたソース・コードを勝手に配布することです．

また，改変したソース・コードの配布もまったく不可能ではありません．このあたりは技術的な話題を外れてしまうので，興味ある方はT-Licenseをご覧ください．

● ソース・コードの改変と移植

ところで，T-Kernelはシングル・ソースで複数のCPUに対応していますが，OSという性格上，完全にすべてのソース・コードをハードウェア非依存とすることは不可能です．OSのごく基本的な部分にハードウェアに依存したソース・コードが存在します．T-Kernelを新しいCPUに移植しようとした場合，このハードウェアに依存したソース・コードを変更しなくてはなりません．しかし，ソース・コードを変更したら改変版とみなされてしまうと，単に新しいCPUに移植しただけのT-Kernelが改変版ということになってしまいます．

そこで，T-Kernelのソース・コードは，基本部とハードウェア依存部に厳密に分けられています．T-Licenseでは，T-Kernelソース・コードの基本部が変更されたものを「改変されたソース・コード」とします．ハードウェア依存部のみが変更されたものは「単純移植されたソース・コード」と呼んで「改変されたソース・コード」と区別しています．

「単純移植されたソース・コード」は定められた手順を踏めば，オリジナルのT-Kernelソース・コードに登録することが可能です．

● ソース・コードの入手

T-Kernelのソース・コードは，以下の手順で申し込みを行い，入手することができます．

1) T-EngineフォーラムのWebページから「T-Kernel利用申し込み」を選ぶ
2) T-Kernel利用申し込みのページで，説明をよく読んだら，ページ下の「利用申込み」のボタンを押す
3) T-License［T-Kernelのソース・コードのライセンス契約］の全文が表示されるのでよく読む．このライセンスに同意し，承諾する場合はページ下の「承諾」ボタンを押す
4) T-Kernel利用を申し込むのが，個人か法人かを選択する．個人の場合は申込者のみがT-Kernel利用の対象となる．法人の場合は属している人達が対象になる
5) 必要な事項を入力する．入力に問題があると許可されないので注意すること
6) 入力した内容を確認する．内容にまちがえがなければ「この内容で申請する」ボタンを押す．T-Engineフォーラムで利用者登録が終わり次第，ダウンロード・ページのURLとユーザID，パスワードが電子メールで送られてくる

以上の手続きが済めば，あとは連絡のあったURLからダウンロードするだけです．

この執筆段階でダウンロード・ページには，T-Kernel 1.00.00版のソース・コードと説明書，および各CPUごとの実装仕様書が置かれています．実装仕様書は，ハードウェアに依存した仕様や実装が記述されています．

● ソース・コードの概要

T-Kernelのソース・コードは，`tkernel.1.00.00.tar.gz`というアーカイブ・ファイルになっています．これを展開すると，`tkernel_source`というディレクトリができあがります．このディレクトリ中には表8に示すディレクトリがあります．

`kernel`ディレクトリにT-Kernel本体のソース・コードが入っています．

`lib`ディレクトリは，T-Kernelが提供するライブラリのソース・コードです．この中にシステム・コールのインターフェース・ライブラリも含まれます．

`include`ディレクトリにはC言語のヘッダ・ファイルが入っています．このファイルは，`kernel`や`lib`中のファイルやユーザ・プログラムから使用されます．

`config`ディレクトリには，`rominfo`，`SYSCONF`，`DEVCONF`などT-Engineのシステムの設定ファイルが入っています．これらの設定ファイルの内容は，各T-Engineの実装仕様書に記載されています．

`etc`ディレクトリにはソース・コードはありません．`make`ルールやスクリプト・ファイルなど，T-Kernelを構築するうえで使用するファイルが入っています．

T-Kernel本体である，`kernel`ディレクトリの構成を図5に示します．`kernel`ディレクトリ以下は機能単位で階層構造になっています．`src`と`build`という名称のディレクトリはそれぞれソース・ディレクトリと構築作業用ディレクトリです．`build`ディレクトリ下にはさらにT-Kernelを構築するターゲット・システムごとの構築作業用ディレクトリがあります．これをカレントとして`make`を実行するとその機能単位がコンパイルされ，ターゲット・システム用のオブジェクトが作られます．

表8　T-Kernelソース・コードの構成ディレクトリ

ディレクトリ名	内　容
kernel	T-Kernel本体
lib	ライブラリ
include	各種定義ファイル（ヘッダ・ファイル）
config	rominfo，SYSCONF，DEVCONFなど設定ファイル
etc	makeルール，スクリプトなど

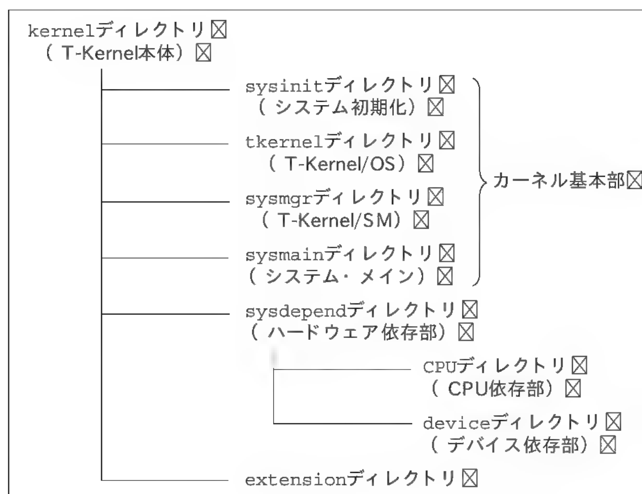


図5 T-Kernelソース・コードのディレクトリ構成図

T-Kernel全体のオブジェクトを作るには，kernel/sysmain/build下のターゲット・システムの構築作業用ディレクトリでmakeを実行します．T-Kernel構築の手順は後述します．

● 基本部とハードウェア依存部

前述のとおりT-Kernelのソース・コードは，カーネル基本部とハードウェア依存部に分けられています．これはディレクトリ単位で厳密に分けられています．

ハードウェア依存部には，現在T-Kernelが対応している各種T-Engineのハードウェアに依存したソース・コードがあります．もしT-Kernelを新しいハードウェアに移植したい場合は，このハードウェア依存部のみをそのハードウェアに応じて書き換えるだけでT-Kernelを移植することができます．

● ユーザ・プログラム

公開されているT-Kernelのソース・コードは，文字どおり本当にT-Kernelだけのソース・コードです．もし，このT-Kernelのオブジェクトを構築して実行したとしても何も意味のある動作は行われません．T-Kernel上で何か意味のあることを行うためには，まず目的のユーザ・プログラムを作成して動作するようにしなくてはなりません．

実際には，T-Kernelのソース・コードには，起動時にシリアル・インターフェースに対して起動メッセージを送り，コンソールからのキー入力を待ってシステムを終了する，という簡単なプログラムがユーザ・プログラムの代わりに書かれています．したがって，この部分をユーザ・プログラムに書き換える必要があります．

● T-Kernelの起動処理

T-Kernelのソース・コードを見ながらT-Engienのシステム起動からユーザ・プログラムが実行されるまでの流れを説明します．これを理解することは，ユーザ・プログラムを書き換えたり，T-Kernelを独自のハードウェアに移植する際に有用だと思います．

T-Engineに電源が投入されると，まず最初にT-Monitorが起動します．T-Monitorは起動時に必要なハードウェアの初期化を行った後，システムの起動処理を開始します．T-Monitorによるシステムの起動処理は，ハードウェアにも依存しますが，基本的な流れは図6に示すとおりです．もしT-Engine上に起動ディスクがあれば，T-Monitorはそこからシステムを起動しようとします．ディスク上にシステムがない場合はROM上の起動プログラムを呼び出します．ROM上の起動プログラムのアドレスは，RomInfoという情報テーブルの中に設定され，決められたROMアドレス上に書き込まれています．RomInfo自体のアドレスはシステムとして固定です．RomInfoのソース・コードは，以下のパスのファイルに記述されています．パス中の<ターゲットシステム名>はT-Engineの種類により変わります．表9を参考に適切な名前に置き換えてください．

/tkernel_source/config/src/sysdepend/<ターゲットシステム名>/rominfo.c

T-Monitorから呼び出される起動プログラムの仕事は一言で言えば，T-Kernelをメモリ上に配置し，T-Kernelの開始アドレスを実行することです．T-Kernelの起動アドレスは以下のファイル中のSTARTです．

/tkernel_source/kernel/sysdepend/device/<ターゲットシステム名>/icrt0.S

もし，T-KernelがROM上に存在し，直に起動する場合は，特別な起動プログラムは不要です．公開されているT-Kernelのソース・コードをそのまま構築するとこの形式となります．よってT-MonitorからT-Kernelの開始アドレスが直接呼ばれ

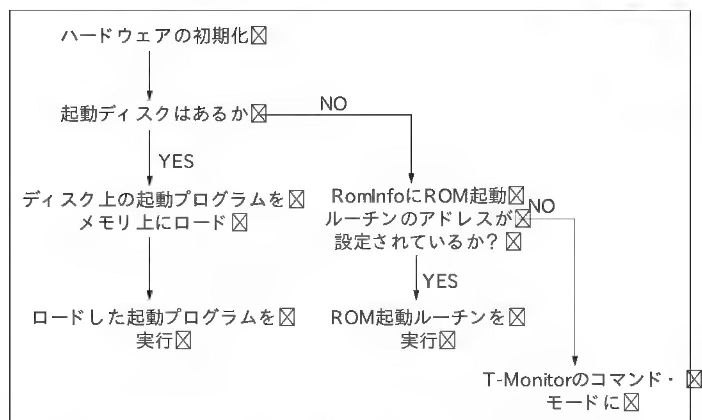


図6 T-Monitorによるシステム起動の流れ図

表9 ターゲット・ディレクトリ名

T-Engine種別	ターゲット・ディレクトリ名
標準T-Engine/SH7727	std_sh7727
標準T-Engine/SH7751R	std_sh7751r
標準T-Engine/V_R5500	std_vr5500
標準T-Engine/ARM720-S1C	std_s1c38k
標準T-Engine/ARM920-MX1	std_mc9328
μT-Engine/M32104	mic_m32104
μT-Engine/V_R4131	mic_vr4131

表10　tm_putstringとtm_getcharの仕様

tm_putstring　コンソールへ1行出力	
書式：	INT tm_putstring(UB *buff);
機能：	buffで指定されたメモリ・アドレスから1文字(1バイト)ずつ，NULLコード(0)までの文字を，デバッグ用コンソール(シリアルI/F)へ出力する． 文字列中のLFコード(0x0A)に対しては，CRコード(0x0D)とLFコード(0x0A)の2文字を出力する．
tm_getchar　コンソールから1文字入力	
書式：	INT tm_getchar(INT wait);
機能：	デバッグ用コンソール(シリアルI/F)から1文字(1バイト)を入力し，入力された値を戻り値として返す． 入力がない場合，waitが0ならば戻り値に−1を返す． waitが0以外ならば入力があるまで待つ．

ます．起動プログラムを独自に用意することで，T-Kernelをディスクからブートしたり，ROMからRAMに転送して起動したり，システムに応じた起動と動作環境を作ることができます．

起動プログラム，またはT-Monitorから直接，T-Kernelが開始されると，T-Kernelはまず内部の初期化処理を実行します．初期化処理が終わると，初期タスク(init_task)と呼ばれる特別なタスクを生成して実行を開始します．

初期タスクが動作している状態では，基本的なT-Kernelの機能はほとんど使用できます．初期タスクの最初の仕事は，タスク・コンテキストで実行する必要のある各種の初期化処理です．初期タスクによる初期化処理が終わるとT-Kernelの初期化処理はすべて終了です．初期タスクは最後にユーザの作成したプログラムを呼び出します．

初期タスクから呼び出されるユーザ・プログラムは，ソース・コード中で初期タスク・メイン処理usermainとして定義されています．このusermainプログラムが，ユーザ・プログラム全体の開始処理を行います．

T-Kernelのソース・コードの以下のパスのファイルを見てください．

/tkernel_source/kernel/usermain/usermain.c

このusermain.cファイルには，usermain()という関数が一つ定義されています．これがusermainプログラムの実体です．

公開されているT-Kernelソース・コードでは，usermain()の中身はたった3行です．

この中ではまずtm_putstring()とtm_getchar()という二つの関数が呼ばれています．この二つの関数は，T-Monitorのサービス関数です．前者はコンソールへの文字列出力を行い，後者はコンソールからの一文字入力を実行します(表10)．つまり，この二つの関数を使って，始めに述べた「シリアル・インターフェースに対して起動メッセージを送り，コンソールからのキー入力を待つ」という処理を行っているわけです．

システムの正常終了は，usermain()関数が戻り値0で終了した際に実行されます．システム終了がどのように行われるかはハードウェアにも依存します．電源をソフトウェアでオフにできるT-Engineは，ここで電源が切られます．

注意が必要なのは，ここで使用されているtm_putstring()やtm_getchar()といったT-Monitorのサービス関数は本来デバッグなどで使用されるものです．というのは，T-Monitorのサービス関数の実行中は，T-Kernelの動作がロックされてしまうため，実際のアプリケーション上で使用するとリアルタイム性などに悪影響を与える可能性があるからです．アプリケーションからシリアル・インターフェースを使用するときは，通常シリアル・インターフェースのデバイス・ドライバを使用します．T-Kernelのソース・コード中でデバイス・ドライバではなく，T-Monitorが使用されているのは，ソース・コードにデバイス・ドライバが含まれていないからです．T-Monitorのサービス関数はちょっとした実験やデバッグには便利なものですがアプリケーションでの使用には十分注意する必要があります．

以上が，T-Engienのシステム起動からユーザ・プログラムが実行されるまでの流れです．ここまでの説明でわかったと思いますが，自分が動かしたいユーザ・プログラムは，このusermain()関数の内容を書き換えることによって実行することができます．

一番単純な方法は，usermain()関数の中にユーザ・プログラム自体を書いてしまうことです．

ただし，より一般的な方法は，usermain()関数の中にユーザ・プログラムを記述するのではなく，usermain()関数からユーザ・プログラムを構成するタスクを生成して起動します．usermain()関数は最初に動くユーザ・プログラムのタスクや，せいぜいそのタスクが使用するカーネル・オブジェクトを生成するくらいに留めたほうが，OSとユーザ・プログラムを明確に区別できてよいでしょう．

なお，usermain()関数が終了するとT-Kernelの終了処理が始まり，システム全体が終了してしまうことに注意してください．システムの終了までusermain()関数を終わらせないように記述する必要があります．もしシステムを終了する必要がないのであれば，初期タスクをスリープさせてしまうことも可能です．

● T-Kernelの構築

T-Kernelのソース・コードの説明が一通り終わったところで，いよいよソース・コードをコンパイル，リンクしてカーネルのオブジェクトを構築してみましょう．これにはソース・コード以外に開発環境と実行環境が必要です．開発環境については，ソース・コードはGCCで書かれているので，GNUのクロス開発環境が必要です．実行環境はもちろんT-Engineです．この開発環境と実行環境の二つは，前に紹介したT-Engine開発キットを入手することによりいっぺんに手に入れることができます．

もっとも，開発キットのT-Engineにははじめから T-Kernelのオブジェクトが含まれているので，単にT-Kernelを動かすだけならわざわざ自分で構築する必要はありません．さらに開

発キットのほうには，デバイス・ドライバやミドルウェアも含まれているので，これを公開されているT-Kernelに置き換えることは機能的にはマイナスです．まずはそのあたりをよく考える必要はあります．もっとも，後から元の状態に戻すことは可能なので，心配しすぎる必要もありません．

以下に，T-Kernelのオブジェクトの構築手順を示します．コマンドの入力例は，開発環境のShellとしてbashを想定しています．%はShellのプロンプトなので入力する必要はありません．ターゲットとするT-Engienは標準T-Engine/SH7727を例にしましたが，異なったターゲットの場合は表9のターゲット・ディレクトリ名を参考に置き換えてください．

1) ソース・コードを適当なディレクトリに展開し，環境変数BDにソース・コードを展開したディレクトリのパス名を設定する．この環境変数BDは開発環境のベース・ディレクトリを示している

```
% export BD=~/tkernel_source
```

2) ライブラリを作る．lib/buildディレクトリ下のターゲット・システムのディレクトリに移動し，makeを実行すると，ライブラリは生成される

```
% cd ~/tkernel_source/lib/build/std_sh7727
% make
```

3) T-Kernel本体を構築する．kernel/sysmain/buildディレクトリ下のターゲット・システムのディレクトリに移動し，makeを実行すると，カレント・ディレクトリ中にkernel-rom.motという名前のファイルでT-Kernel本体のオブジェクト・ファイルが生成される

```
% cd ~/tkernel_source/kernel/sysmain/build
                              /std_sh7727
% make
```

4) RomInfoのオブジェクトを作成する．config/buildディレクトリ下のターゲット・システムのディレクトリに移動し，makeを実行すると，カレント・ディレクトリ中にrominfo.motという名前でRomInfoのオブジェクト・ファイルが生成される

```
% cd ~/tkernel_source/config/build/
                                std_sh7727
% make
```

以上でカーネルの構築は終了です．

● T-Kernelの実行

構築したT-KernelのオブジェクトをT-Engineで動かすには，まずオブジェクト・ファイルkernel-rom.motをT-Engineのフラッシュ・メモリに書き込みます．フラッシュ・メモリの書き込みは，通常T-MonitorのFlashLoad(FLLO)コマンドで行うことができます．フラッシュ・メモリへの書き込み方法は，T-Engineの種類やバージョンによっても違う場合があるので，T-Engineに付属のマニュアルを読んでください．

フラッシュ・メモリへの書き込みを行うと，それまでフラッシュ・メモリにあった開発キットのソフトウェアが上書きされてしまうので注意してください．元のフラッシュ・メモリのソフトウェアに戻すには，開発キットに付属しているROMイメージのオブジェクト・ファイルを書き戻します．この方法も，各T-Engineに付属のマニュアルに記載されているので，そちらを読んでください．

kernel-rom.motの書き込みが終了したら，続いてRomInfoのオブジェクト・ファイルrominfo.motを同様の手順でフラッシュ・メモリへ書き込みます．このRomInfoにはシステムの起動に必要な情報が設定されているので，システムのソフトウェアを変更した場合はいっしょに変更が必要です．

二つのファイルのフラッシュ・メモリへの書き込みが終わってから，T-Engineの電源を入れなおすと，新たに書き込んだT-Kernelが起動します．この際，T-Engineのシリアル・インターフェースにパソコンを接続し，ターミナル・ソフトを動かしておくと，T-Kernelからの出力が表示されます．通信設定はT-Monitorとの通信の設定と同じです．

構築したT-Kernelが正常に起動すれば，以下のメッセージが表示されるはずです．

```
T-Kernel Version 1.00.00
Push any key to shutdown the T-Kernel.
```

ここで，パソコンのターミナル・ソフトから何らかのキー入力を送ると，T-Kernelは終了します．T-Engineのハードウェアにもよりますが，T-Monitorから電源を切る機能のあるT-Engineならば，ここで電源を切ることもできます．

以上でT-Kernelの構築から起動までは終わりです．

おわりに

T-Kernelについて，その特徴から公開されているソース・コードを用いてT-Kernelを構築する方法までを説明してきました．少々駆け足となりましたが，T-KernelはどのようなOSか，またμITRONとはどのように違うのか，それを理解するうえでの糸口となればと思います．

T-Kernel自体はまだ生まれたばかりのOSですが，T-Engineプロジェクトとともに急速に発展し，広まってきています．これからさらに多くのCPUにT-Kernelが対応し，また多くのT-Kernelを用いた製品が世に出てくると思います．

T-Kernelのソース・コードはだれでも手にすることができます．T-Kernelに少しでも興味もたれた方は，ぜひ実際に使ってみてください．

参考文献
(1) T-Kernel標準ハンドブック，坂村 健 監修/T-Engineフォーラム 編著，パーソナルメディア(株)
(2) T-Engineフォーラム，http://www.t-engine.org/
(3) TRONプロジェクト，http://www.tron.org/

とよやま・ゆういち
YRPユビキタスネットワーキング研究所 基盤システム研究室

TRONSHOW 2004で注目を集めた

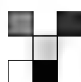
第4章 T-Engine開発キットとTeacube

松為 彰

T-Engineで開発を行うためには，開発用のT-EngineボードとT-Kernel，そしてコンパイラなどの開発環境が必要となる．これらを自分で揃えることも可能だが，すでにこれらをセットにした開発キットが供給されている．本章ではまず，この開発キットについて解説する．

また，T-Engineの応用製品として昨年末のTRONSHOW 2004では“Teacube（旧名称：T-Cube）”が公開され，注目を集めた．「みかん大のパソコン」と紹介されることも多いTeacubeだが，開発者にとってはT-Engine応用プラットホームとしての意義も大きい．そこで，Teacubeについてもあわせて解説する．

（編集部）



T-Engine開発キットとは

T-Engineベースのシステムを開発したり，T-Engine用のミドルウェアを開発する際に利用できる便利な製品が「T-Engine開発キット」である．T-Engine開発キット[1]には，T-EngineやμT-Engineのボード（ハードウェア）のほか，その上で動くT-MonitorとリアルタイムOSのT-Kernel，T-Engineボードの周辺機器をサポートするデバイス・ドライバ，ファイル管理などの機能をもつT-Kernel Extensionという基本ミドルウェア，Linux上で動作するGNUベースの開発環境，接続用のシリアル・ケーブルなどを含んでおり，T-Engine関連のシステム開発に最低限必要となるハードウェアとソフトウェアを一通りそろえたパッケージ製品である．Linuxの動く開発用ホストPC[注1]と，T-Engine開発キットがあれば，T-Engineのソフトウェア開発が可能である．

T-Engine開発キットには，CPU別に多くの種類がある（表1）．基本的な製品構成はいずれも共通だが，付属するボード類や一部のミドルウェアについては，CPUの特性などが反映されているので差異がある．たとえば，LCDボードについては，T-Engine開発キットに含まれている場合（T-Engine/SH7727開発キットなど）と，そうでない場合（T-Engine/V_R5500開発キットなど）がある．LCDボードが付属しないT-Engine開発キットでは，オプション製品としてLCDボードを

表1　発売中のT-Engine開発キット

機種名	CPUアーキテクチャ	LCDボード	デバッグ・ボード	標準価格（税込）	備　考
T-Engine/SH7727開発キット	SH3-DSP（ルネサス）	付属	付属	￥152,250	
T-Engine/SH7751R開発キット	SH4（ルネサス）	付属	付属	￥204,750	
T-Engine/SH7760開発キット	SH4（ルネサス）	付属	付属	￥204,750	I/Oボード付属
T-Engine/V_R5500開発キット	MIPS（NECエレクトロニクス）	オプション	N-Wireコネクタあり	￥207,900	コネクタ・ボード付属，アナログRGB出力あり
T-Engine/TX4956開発キット	MIPS（東芝）	オプション	N-Wireコネクタあり	￥207,900	コネクタ・ボード付属，アナログRGB出力あり
T-Engine/ARM720-S1C開発キット	ARM7（EPSON）	オプション	オプション	￥207,900	
T-Engine/ARM920-MX1開発キット	ARM9（Motorola）	オプション	オプション	￥207,900	
T-Engine/ARM720-LH7開発キット	ARM7（SHARP）	オプション	オプション	￥207,900	
T-Engine/ARM922-LH7開発キット	ARM9（SHARP）	オプション	オプション	￥207,900	
T-Engine/ARM926-MB8開発キット	ARM9（富士通）	オプション	オプション	￥260,400	ETMコネクタあり

(a) 標準T-Engine

機種名	CPUアーキテクチャ	MMU対応とT-Kernel Extension	標準価格（税込）	備　考
μT-Engine/M32104開発キット	M32R（ルネサス）	なし	￥157,500	LANボード，ARボード付属
μT-Engine/V_R4131開発キット	MIPS（NECエレクトロニクス）	付属	￥165,900	N-Wireコネクタあり

（2004年5月末現在）

(b) μT-Engine

注1：後述のように，Cygwinの併用によってWindows上で開発することも可能である．

供給している(LCDインターフェースを持たないμT-Engineを除く).一方，T-Engine/V_R5500開発キットやT-Engine/TX4956開発キットでは，CPUボードがアナログRGB出力の機能をもっており，PC用のCRTディスプレイを接続できるため，15ピンD-SUBのCRTコネクタを搭載した接続用のコネクタ・ボードが付属している．また，ソフトウェアに関しては，μT-Engineの一部の機種でT-Kernel Extensionが付属していない．これは，CPUがMMU(Memory Management Unit)をサポートしておらず，T-Kernel Extensionを実装できないためである．

T-Engine開発キット(表2)には，ドキュメントおよびソフトウェアを収録したCD-ROMが付属している(図1)．CD-ROMには，開発キット全体の取扱説明書のほか，ハードウェアの仕様書および回路図，T-MonitorおよびT-Kernelの仕様書，T-MonitorやT-KernelのCPU依存部分の仕様を記述したCPU別の実装仕様書，デバイス・ドライバおよびライブラリの説明書，T-Kernel Extension説明書，GNU開発環境説明書などの豊富なドキュメントがPDFファイルで収録されている．なお，ハードウェアの仕様書および回路図は，機種によっては別のCD-ROMに収録されている．

一方，CD-ROMに含まれるソフトウェアとしては，実行用のT-EngineやμT-Engine上で動作するターゲット側ソフトウェアと，開発用のホストPCで動作するホスト側ソフトウェア(GNU開発環境)がある．ターゲット側ソフトウェアの具体的な内容は，T-Monitor，T-Kernel，T-Kernel Extension，デバイス・ドライバの実行用バイナリ・イメージである．T-Engine開発キットでは，原則として出荷時にこれらのソフトウェアはフラッシュ・メモリにプリインストールされているが，何らかの理由で再インストールする必要が生じたり，これらのソフトウェアのバージョン・アップがあった場合には，ユーザ側でフラッシュ・メモリへの再インストールを行う．フラッシュ・メモリの書き換えにはT-Monitorの機能を使うので，このとき，T-Monitorは自分自身を書き換えることになるので注意が必要だ[注2]．

開発キットには，このほか，ファイルの内容比較(cmp)や簡易行エディタ(ed)など，ユーティリティとしても利用可能なサンプル・プログラムがいくつか付属している．これらのサンプル・プログラムとデバイス・ドライバの一部については，ソースも含めて提供されている．

表2　T-Engine開発キットの製品パッケージの内容(SH7727版の場合)

ハードウェア
● SH7727を搭載した標準T-Engine仕様のCPUボード(モデル番号h101) ● LCDボード(タッチ・パネル付き) ● デバッグ・ボード(フラッシュ・メモリ書き込み用およびH-UDIデバッガ接続用) ● ACアダプタ ● RS-232-C(シリアル)ケーブル
ターゲット用ソフトウェア
● T-Monitor ● T-Kernel ● T-Kernel Extension(開発用基本ミドルウェア) 　ファイル管理機能，CLI(コマンド・ライン・インタプリタ)など ● PCカード・マネージャ(バス・ドライバ)，USBマネージャ(バス・ドライバ) ● デバイス・ドライバ：ソース・プログラム付き 　時計(RTC)，コンソール(シリアル)，スクリーン(LCD)，システム・ディスク(ATAカード，USB)，KB/PD(キーボード，タッチ・パネル，マウス) ● サンプル・アプリケーション：ソース・プログラム付き 　簡易ディスク区画作成(hdpart)，ディスク・フォーマット(format)，ファイルの内容比較(cmp)，ディスク・ダンプ(dd)，簡易行エディタ(ed)
開発マシン用ソフトウェア
● GNU開発環境：ソース・プログラム付き 　PC-Linux上で動作するGNUベースの開発環境(ソース/デバッガgdbを含む) 　Cygwinを利用することで，Windows上でも開発が可能
ドキュメント(CD-ROM内の電子ファイルで提供)
● 開発キット取扱説明書 ● ライブラリ説明書 ● デバイス・ドライバ説明書 ● T-Kernel Extension説明書 ● GNU開発環境説明書 ● GNU開発環境(Windows版)説明書 ● T-Monitor仕様書 ● T-Kernel仕様書 ● T-Monitor/T-Kernel実装仕様書(SH7727版) ● ハードウェア取扱説明書(回路図を含む)

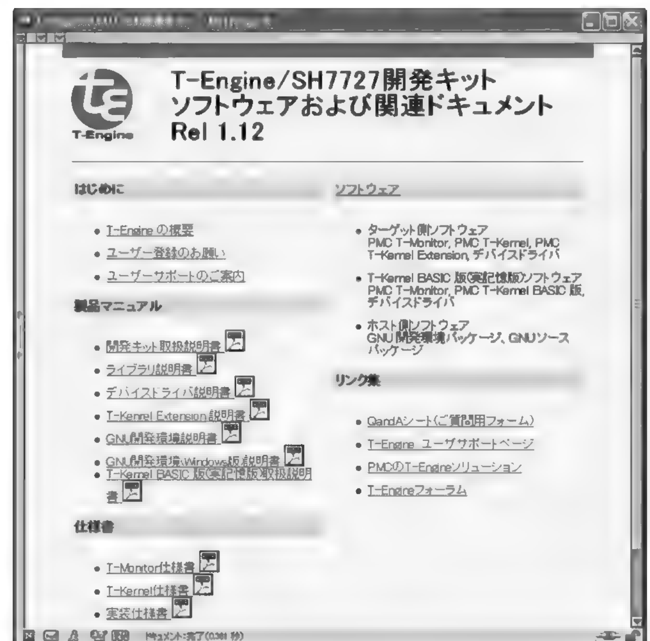
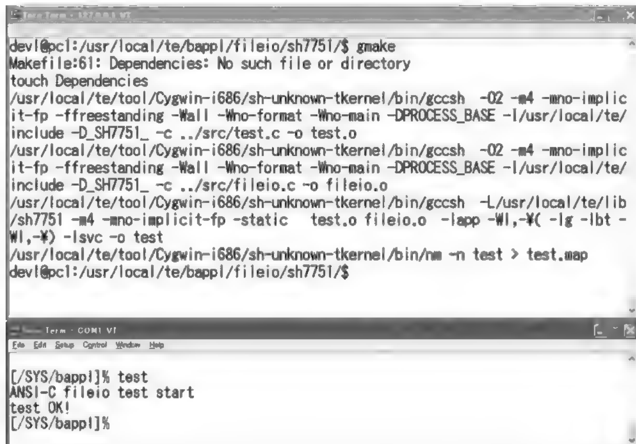


図1　T-Engine/SH7727開発キットに付属しているCD-ROM

注2：この書き換えに失敗すると，何らかのハードウェア的な手段を使って，フラッシュ・メモリへの再インストールを行う必要が生じる．具体的な方法は機種に依存しており，SH7727，SH7751R，SH7760の場合は付属のデバッグ・ボードを接続すればフラッシュ・メモリへの再書き込みが可能である．ほかの機種では，ICEを接続してこの作業を行う．


```
devl@pcl:/usr/local/te/bappl/fileio/sh7751/$ gmake
Makefile:61: Dependencies: No such file or directory
touch Dependencies
/usr/local/te/tool/Cygwin-i686/sh-unknown-tkernel/bin/gccsh  -O2 -m4 -mno-implic
it-fp -ffreestanding -Wall -Wno-format -Wno-main -DPROCESS_BASE -I/usr/local/te/
include -D_SH7751_ -c ../src/test.c -o test.o
/usr/local/te/tool/Cygwin-i686/sh-unknown-tkernel/bin/gccsh  -O2 -m4 -mno-implic
it-fp -ffreestanding -Wall -Wno-format -Wno-main -DPROCESS_BASE -I/usr/local/te/
include -D_SH7751_ -c ../src/fileio.c -o fileio.o
/usr/local/te/tool/Cygwin-i686/sh-unknown-tkernel/bin/gccsh  -L/usr/local/te/lib
/sh7751 -m4 -mno-implicit-fp -static   test.o fileio.o  -lapp -Wl,-\( -lg -lbt -
Wl,-\) -lsvc -o test
/usr/local/te/tool/Cygwin-i686/sh-unknown-tkernel/bin/nm -n test > test.map
devl@pcl:/usr/local/te/bappl/fileio/sh7751/$

[/SYS/bappl]% test
ANSI-C fileio test start
test OK!
[/SYS/bappl]%
```

図2　Cygwinにより Windows 上で実行中のT-Engine開発環境

ちなみに，T-EngineやμT-Engineのボードを製造しているのは，それぞれのCPUメーカや関連メーカだが，販売ルートなどの問題もあり，一般ユーザがこれらのボードを単品で購入することは難しい場合が多い．また，これらのボードに合わせてGNUの開発環境を調整したり，T-Engineフォーラムから提供されるT-Kernelをこれらのボードに移植して周辺の環境を整える作業には，かなりの手間がかかる．だが，T-Engine開発キットを購入すれば，こういった手間がかからず，すぐにT-Engine用ミドルウェアやデバイス・ドライバ，アプリケーションなどの開発を始めることができ，開発期間短縮や開発コスト削減の効果が大きい．

T-Engine開発キット付属のクロス開発環境

T-Engine開発キットには，Linux上で動くGNUベースのクロス開発環境が付属している．この中には，GNUベースのソース・レベル・デバッガであるgdbも含まれており，シンボル名を使って変数の参照や値の設定を行ったり，ソース上でブレーク・ポイントを設定することができる．この開発環境は，Red Hat Linux 7.1/7.3/8.0/9.0 および Red Hat Professional Workstation 上での動作を確認しているほか，UNIXをエミュレーションする環境であるCygwinを併用することにより，Windows上で動作させることもできる(図2)．

また，ターゲットのT-Engine上で動作するユーティリティとして，IMS(Initial Monitor System)，フォーマッタ，CLI(Command Line Interpreter)などのツール類が提供されている 注3．T-Engineのシリアル・ポートに通信端末を接続し，T-Engine上でこれらのツールを実行することにより，タスクやセマフォなどのオブジェクトやファイルの状態を参照/設定したり，開発したプログラムのダウンロードや実行，デバッグなどを行うことができる．なお，標準T-EngineのCPUボードには，ハードディスクなどのストレージ・デバイスが付いていない．そのため，一般には，作業用ディスクとしてPCMCIAスロットに挿入されたCF(Compact Flash)を利用し，開発システムからダウンロードしたデバッグ中のプログラムやミドルウェア，OS本体，各種の設定状態などのファイルをこの中に格納する 注4．

このほか，USB経由でハードディスクやSDカード・リーダなどを接続し，それらのデバイスを作業用ディスクとすることもできる．また，T-Engineボードに搭載しているフラッシュ・メモリ上にファイル・システムを構築し，この中にプログラムなどを格納することも可能である．ただし，フラッシュ・メモリの場合にはファイル単位での書き換えができず，フラッシュ・メモリ全体を一括して書き換える必要があるため,この上のファイル・システムはリード・オンリで利用する．したがって，デバッグ中のプログラムや書き換えの必要なデータ類をフラッシュ・メモリに格納することはできないが，開発済みのアプリケーションやOS，ミドルウェアなどを格納する目的であれば，T-Engine上のフラッシュ・メモリを活用できる．

T-Engine開発キットのオプション製品

T-Engine開発キットには，各種の便利なオプション製品も用意されており，T-Engine開発キットといっしょにパーソナルメディアから購入できる(1)．オプション製品には，対応機種(CPU)専用の製品もあるし，一部の機種の間で共通に使えるものもある．これらのオプション製品について紹介する．

なお，本稿ではハードウェアのオプション製品について説明しているが，このほか，T-Engine開発キット上で動くミドルウェア類も開発キットのオプション製品として販売されている．ミドルウェアについては第5章を参照していただきたい．

● T-Engine開発ベンチ

T-Engine開発キットのCPUボードやLCDボードを保護するアクリル製の台やカバーと，LCDボード用のボタン・スイッチ，四隅を止める支柱やネジなど，周辺の治具類をセットにした製品である．T-Engineのボード類は，むき出し状態のままで利用することも可能だが,デモや実験のための持ち運び，取り扱いのしやすさなどを考えると，ある程度物理的に保護しておくほうが望ましい．

注3：一部のμT-Engine開発キットでは，CPUがMMUを持たないため，T-Kernel Extensionが実装されておらず，CLIも付属しない．この場合には，ファイル・システムも利用できない．

注4：μT-Engineの場合は，PCMCIAスロットの代わりにCFスロットがあるので，やはりCFを利用する．ただし，USBは利用できない．また，前述のように，一部のμT-Engineではファイル・システムを利用できない．

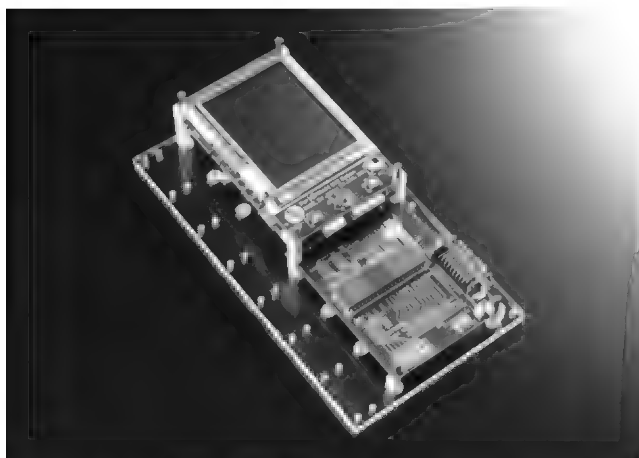


写真1　T-Engine開発ベンチ(アクリル台)



写真2　T-Engine開発ベンチ(携帯用カバー)

T-Engine開発ベンチは，この目的で開発された商品であり，付属しているアクリル板の使い方の組み合わせによって，厚めのアクリル板を据え置き式の台として利用したり(写真1)，アクリル製のカバーを携帯用の簡易式ケースとして利用するなど(写真2)，いろいろな使い方が可能である．

アクリル製のカバーは，標準T-EngineのCPUボード，および標準T-Engine用のLCDボードのサイズにちょうど合った大きさになっている．また，拡張ボードを追加した場合にも，支柱を増やし，ネジを長くすることで対応できる．一方，据え置き式のアクリル台は，デバッグ用ボードなど，CPUボードとは重ならない方向に飛び出した拡張ボード類や，μT-EngineのCPUボードにも使えるような設計になっている．アクリル板は透明なので，ディップ・スイッチなどボード上の各部品の状態を確認しやすい．

T-Engineでは，ボードのサイズだけではなく，ネジ穴やコネクタ類の位置も標準化されているため，機種(CPU)が変わっても，アクリル製の台やカバーやネジ類はすべて共通に利用できる．そのため，本製品はすべての標準T-Engineに共通して利用可能なオプションとなっている．T-Engine開発ベンチは，CPUボードの物理的な仕様をも標準化するというT-Engineプロジェクトの特性を最大限に生かした製品だといえるだろう．

● T-Engine/μT-Engine拡張バス専用コネクタ

T-EngineおよびμT-EngineのCPUボードには，京セラエルコ社がT-Engineプロジェクトのために新規開発した拡張バス・コネクタ[2]が付いており，このコネクタ経由でユーザの開発した拡張ボードを接続できる．その際に必要なバス・コネクタも，T-Engine開発キットのオプションとして購入できる．

ちなみに，T-Engine(μT-Engineを含む)では，実行効率重視の観点から拡張バスの仕様[注5]を標準化していない．このため，別のCPUボードのために開発された互換性のない拡張ボードを誤って挿入しないように，CPUボードごとにコネクタの切り欠き部分のパターンを変え(一部の機種では共通)，コネクタ挿入の可否を論理的に区別できるようになっている．この切り欠きパターンをキーイングと呼んでいる．バス・コネクタ購入の際には，プラグ/リセプタクル(オス/メス)の指定とともに，キーイング(対応機種)を指定する必要がある．これらのコネクタに関しては特集Appendix 1を参照してほしい．

● LCDボード

標準T-EngineのCPUボードには，LCDボードとの接続インターフェースが用意されており，QVGAのタッチ・パネル付きLCDボードを接続できる．また，このLCDボードには，"○","×"のボタン，および十字カーソル機能をもつボタンが付いており，ちょっとしたHMIの構築や実験に利用できる．T-Kernel Extensionのイベント管理機能を利用すると，LCDボード上のこれらのボタンと，USB接続したキーボードのカーソル・キーなどを同じAPIでアクセスできる．

LCDボードは，SH7727版，SH7751R版，SH7760版などの一部の機種ではT-Engine開発キットに付属しているが，ARM版やV_R版などほかの機種ではT-Engine開発キットに付属しておらず，オプション製品として供給される．

● デバッグ・ボード

デバッグ・ボードは，ARM版のT-EngineでICEを接続する場合などに別途必要となる．ただし，「T-Engine/ARM926-MB8開発キット」など一部の機種では，CPUボードにETMコネクタが付いており，これを利用できるICEであればデバッグ・ボードは不要である．また，V_R版のT-Engineでは，CPUボードにN-Wireコネクタが付いているので，デバッグ・ボードは利用しない．

一方，SH7727，SH7751R，SH7760などSH版のT-Engineでは，T-Engine開発キットにデバッグ・ボードが付属している．このデバッグ・ボードは，ICEとの接続に使用するほか，

注5：これらの情報は，T-Engine開発キット付属のドキュメントに記載されている．

ボード上にROMベースで動く簡単なモニタが載っており，通信端末を接続してCPUボード上のフラッシュ・メモリを書き換えるなどの機能をもっている．

デバッグ・ボードの必要性や機能に関しては，上記のように機種によって異なるので，購入時には詳細情報をご確認いただきたい．

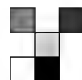
● 拡張LANボード

T-EngineやμT-EngineのCPUボードにはLAN機能を搭載していないので，LANが必要であれば，PCMCIAのLANカード，あるいは拡張LANボードを利用する必要がある．SH7727/SH7760版，SH7751R版，ARM版などのT-Engine開発キットに対して，オプションの拡張LANボードが販売されている．なお，μT-Engine/M32104開発キットに関しては，開発キットに拡張LANボードが付属しているため，オプション製品として別途購入する必要はない．

● 拡張ユニバーサル・ボードとFPGAボード

ユーザが自分でT-Engine用の拡張ボードを開発する場合には，前述のT-Engine/μT-Engine拡張バス専用コネクタを利用すればよいのだが，このコネクタは0.5mmピッチ，140極タイプのものであり，この部分を手作業で配線するのは困難である．そこで，実験などの目的で手作業による拡張ボードの製作もできるように，一部のT-Engineのオプション製品として拡張ユニバーサル・ボードが販売されている．

このほか，T-Engine用のFPGA(Field Programmable Gate Array)ボードの販売も予定されている．FPGAボードの利用により，プロトタイプや最終製品で要求されるハードウェアを，さらに効率よく開発することができる(3)．



T-Engine Appliance評価キットとTeacube

● T-Engine Appliance評価キットとは何か

T-EngineやμT-Engineの本来の位置付けは，開発評価用ボード，あるいはプロトタイプの開発用ボードである．しかし，従来の開発評価用ボードと比較してコンパクトであり，またPCMCIAスロットやUSBコネクタなど汎用性の高い入出力を持っているため，最終製品にそのまま組み込んで利用することも可能である．具体的には，券売機やKIOSK端末，コピー機や複合機といった大きめのOA機器など，物理的にも機能的にもサイズが大きく，かつ多品種少量生産となる場合や，ハードウェアを別途量産するほどのまとまった台数が出ない場合には，開発コストや開発期間の面から，T-EngineやμT-EngineのCPUボードをそのまま最終製品に組み込んで利用するほうが有利な場合もある．ハードウェアを別途開発すると，それ自体の開発費や開発期間がかかるほか，ソフトウェアの移植作業や移植後のテスト，検証作業が必要となるからである．

T-Engineはハードウェアの差異に対してソフトウェアの互換性が極めて高く，移植作業自体の工数はそれほど問題にならないことが多いものの，ハードウェアを新規に開発している以上，ハードウェアとの整合性も含めた検証作業は必要である．ハードウェアを作り直した場合には，不要機能の削除などによる製造単価の削減が期待でき，製造台数が多ければそのメリットが大きくなるので，そのメリットが移植や検証の工数を上回るかどうかがポイントとなる．

ところで，T-EngineのCPUボードをそのままシステムに組み込んで使う場合，もう少し高機能な周辺I/Oを使えると便利な場合が多い．前述のように，T-Engineの本来の目的は開発評価用プラットホームであることから，ハードウェア構成の自由度を優先した設計方針になっており，周辺機器は必要に応じてユーザ(最終製品の開発者)が追加して利用するという考え方になっている．その意味では，T-Engineは必要最小限に近い周辺I/Oしか持っていない．こういった事情から，T-Engineをそのまま最終製品に組み込んで使おうとすると，もう少し高機能な周辺I/Oを求められる場合が出てくるのである．もちろん，必要な周辺I/Oを持つ拡張ボードを追加購入したり，自分で開発すれば，この点は解決できるが，最終製品に対するコストや製品に組み込むための物理サイズといった点で，拡張ボードの利用が難しいケースもある．

こういった背景から，T-Engineよりも最終製品に近い位置付けと機能をもった製品群として，「T-Engine Appliance評価キット」というジャンルが生まれた．T-Engine Appliance評価キットでは，T-Engine開発キットと比較して，LAN機能，高解像度グラフィックスなど，最終製品としても便利に使える機能が追加されている．また，T-Engineでは基板のサイズはもちろん，USBやPCMCIAスロット，拡張バス・コネクタの種類や位置まで決まっていたが，T-Engine Appliance評価キットでは物理サイズや周辺I/Oなどのハードウェア標準仕様には拘束されない．実装者の判断により，さらにコンパクトで高機能なハードウェアにしたり，それを適切な筐体に収めて提供することも可能である．T-Engineのようなむき出しのボードではなく，コンパクトな筐体に入った制御用コンピュータの形状とすることにより，最終製品の具体的なイメージを描きやすくなるといったメリットもある．逆に，T-Engine用の拡張ボードや開発ベンチなど，T-Engineの物理形状に合わせて開発されたハードウェアのオプション製品は利用できない場合がある．

● ソフトウェアは互換性がある

一方，ソフトウェアに関しては，T-Engine開発キットとT-Engine Appliance評価キットで基本的な違いはなく，CPUアーキテクチャが同じ機種であれば，ほとんどのソフトウェアが共通あるいは互換で使える．具体的には，T-Engine Appliance評価キット上でもT-Engine開発キットとほぼ同じT-Monitor，T-Kernel，T-Kernel Extensionなどが動き，ミドルウェアやアプリケーションに関しても，ほぼ完全な互換性がある．LANや高解像度グラフィックスなど，周辺デバイスの機能追加があった



写真3　Teacube/V$_R$5701評価キット本体



写真4　Teacube/V$_R$5701評価キットの試作品とブラウザの実行

部分については，対応するデバイス・ドライバやT-Monitorなどのデバイス依存部分のみが修正されており，上位のミドルウェアやアプリケーションの互換性には影響しない．

T-Engine Appliance評価キットは，上記のように，T-Engineよりも最終製品に近い位置づけを狙っている．とはいえ，組み込み機器のシステム構成には非常に広範囲のバリエーションがあり，特定の機種やシステム構成ですべての要求に対応できるわけではない．その意味で，T-Engine Appliance評価キットも最終的な応用製品そのものではなく，あくまでも半応用製品といった位置づけの「評価キット」であり，これを最終製品として利用するにはユーザ側でのカスタマイズを要するのが普通である．この点はT-Engine開発キットと同様であるが，T-Engine Appliance評価キットの場合，プロトタイプから最終製品までのカスタマイズの量が少なくて済み，開発期間や開発コストをさらに節約できるというメリットがある．

Teacube/V$_R$5701評価キットのハードウェア

上記のようなコンセプトを最初に具体化した製品が，「Teacube/V$_R$5701評価キット」[1]である．本製品は，MIPSアーキテクチャの標準T-EngineであるT-Engine/V$_R$5500開発キットのCPUボードと，LAN機能などを持つそれ用の拡張ボード（PC用拡張ボード）の組み合わせをベースに，さらに集積度を上げて超小型の制御用コンピュータとしたものである．名前のとおり，50mm立方のコンパクトなキューブ型の金属ケースに収められている（写真3，写真4）．本製品の試作品は昨年末に開催されたTRONSHOW 2004で発表され，「超小型PC」として多くのマスコミから注目を集めた後，今年の3月から実際の製品の出荷が始まった．PCといっても，x86系のCPUやWindowsを搭載しているわけではないが，PCと同等の機能，具体的にはWebブラウザをはじめ，ワープロ・ソフト，プレゼンテーション・ソフトなどをMIPS系の組み込み用CPUとTRONのウィンドウ・システムを使って実現しており，PCと同等以上の高度な応用が可能である．

本製品のCPUには，NECエレクトロニクスのV$_R$5701を採用している．このCPUは，組み込み向けの高性能CPUとして開発されたV$_R$5500に，IDEなどの周辺回路を内蔵して集積度を上げたものだが，命令セットなどCPUアーキテクチャは両者同じであり，開発環境（コンパイラ）についてもまったく同じものが使える．Teacube/V$_R$5701の入出力としては，グラフィック画面用のアナログRGB出力とTrue IDEスロット[注6]のほか，USBホストとシリアルが各2ポート，LAN，オーディオ出力，マイク入力端子が付いており，CRTモニタやUSBのマウスおよびキーボードなど，PCと共通の周辺機器が利用できる．グラフィック制御にはSM1722（Lynx3DM+）を採用しており，SXGA（1280×1024ドット）の解像度まで実現できる．USBの制御にはNECのμPD720101，LANの制御にはIntel i82559ERを使っている（表3）．なお，T-Engineと同様に，ハードディスクは積んでおらず，T-KernelなどのOSやシステム・ソフトウェアはフラッシュ・メモリまたはCFに格納する．

注6：物理的にはCFスロットと同じものであるが，ストレージ系のCFにのみ対応しており，活線挿抜はできない．

表3　Teacube/V$_R$5701評価キットとT-Engine/V$_R$5500の入出力の比較

	T-Engine/V$_R$5500＋PCEX*	Teacube/V$_R$5701
CPU	NEC V$_R$5500	NEC V$_R$5701（V$_R$5500コア）
Memory	SDR SDRAM 128Mバイト	DDR SDRAM 64Mバイト
Video/VRAM	Silicon Motion SMI712 /2Mバイト*	Silicon Motion SMI722 /4Mバイト
NIC（LAN）	Intel i82559ER*	Intel i82559ER
USB	NEC VRC5477内蔵（OHCI）	NEC μPD720101（EHCI/OHCI）
IDE I/F	HighPoint HPT372*	NEC V$_R$5701内蔵
PCMCIA I/F	RICOH R5C475II	なし

*PC用拡張ボード（未発売）

Teacube/V$_R$5701評価キットのソフトウェア

Teacube/V$_R$5701評価キットは，ソフトウェアに関しても最終製品に近い高度な機能を提供している．具体的には，T-Engine開発キットに付属しているT-Monitor，T-Kernel，T-Kernel Extensionのほか，GUIやマルチウィンドウ・システム，多漢字をサポートするPMC T-Shellと，その上で動くブラウザなどのアプリケーションが付属している(表4)．このうち，PMC T-Shellの詳細については第5章で説明しているが，Teacubeでは，T-Engine開発キットよりも広いグラフィック画面を使えるため，PMC T-Shellの特性をさらに生かした使い方が可能である．

表4　Teacube/V$_R$5701評価キットに付属しているソフトウェア

OS，ミドルウェアなど
●T-Monitor ●T-Kernel ●T-Kernel Extension(開発用基本ミドルウェア) 　ファイル管理機能，CLI(コマンド・ライン・インタプリタ)など ●デバイス・ドライバ ●PMC T-Shell(ミドルウェア集) 　ディスプレイ・プリミティブ，GUIマネージャ，かな漢字変換機能，17万字の多漢字・多言語用フォント，TCP/IPなど
基本アプリケーション
●基本ブラウザBBB(Web閲覧用ソフト) ●基本文章編集(ワープロ・ソフト) ●基本図形編集(図形編集ソフト) ●マイクロスクリプト(ビジュアル言語) ●システム環境設定 ●ユーザ環境設定 ●ネットワーク設定
開発マシン用ソフトウェア
●GNU開発環境：ソース・プログラム付き 　PC-Linux上で動作するGNUベースの開発環境 　(デバッガgdbを含む)

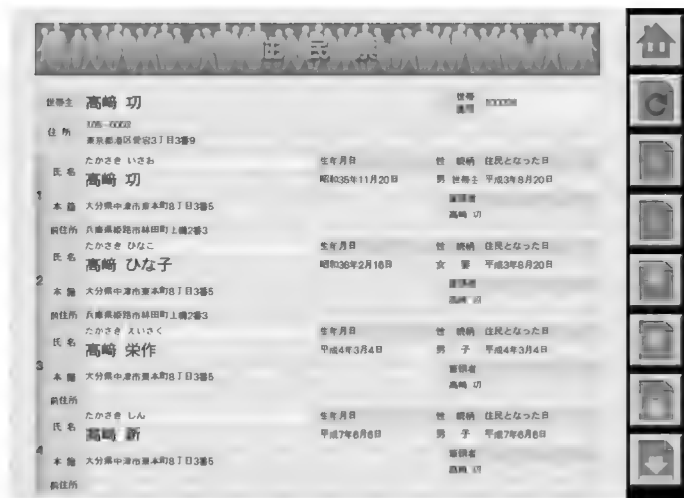


図3　Teacube付属ブラウザ(BBB)のKIOSKモードで実行した住民票端末
人名で使われる異体字も，JISやUnicodeの漢字と同様に利用できる

Teacube/V$_R$5701評価キットで特徴的なソフトウェアは，Teacubeの大きな画面を生かすTRON仕様のWebブラウザ，BBB(BTRON Basic Browser)である．組み込みシステムの高度化にともない，GUIが複雑になると，GUIによる設定内容や画面デザインを定型的なフォームで扱ったり，OSや個々のシステムとは独立して管理，運用したいという要求が出てくる．こういったケースでは，GUIを含めた画面デザインをHTMLで記述し，ブラウザを使ってそれをターゲット・システムの画面上に実現するのも一つの方法である．こういった用途に加えて，業務用端末や電子政府端末など，いわゆるインターネット・アプライアンス系の組み込みシステムを中心に，ブラウザの需要は高い．こういった需要に応える形で，Teacube/V$_R$5701評価キットではBBBを標準添付としている．

BBBで特徴的な機能は，テキスト形式TRONコード(&T形式)[注7]で表現された多漢字に対応しており，PMC T-Shellの多漢字機能と組み合わせて，多漢字の表示や入力ができることである．この機能は，正確な人名用漢字を多数扱うべき電子政府向けシステムや会員情報管理システムはもちろん，書名や著者名に多漢字が必要な図書館の蔵書データベース・システムなど，幅広い用途で役立つ．

また，BBBには，博物館のガイダンス・システムを実現するために開発されたKIOSKモードの機能がある．KIOSKモードでは，画面のクリックによるリンク先へのジャンプやスクロールの操作は可能だが，ブラウザが全画面モードになり，ウィンドウ枠も消えてしまうため，クリックやダブルクリックでBBBのアプリケーションを終了することは不能になる(マウスの右ボタンでメニューを出して終了することは可能)．そのため，タッチ・パネルなどでマウスの左ボタンに相当する機能のみを生かしておけば，ユーザの故意または過失でBBBやシステムを終了したり，ほかのアプリケーションを実行するという危険がなくなる．また，操作が一定時間行われないと，自動的にトップ・ページに戻る機能もある．Teacubeでは，BBBのこういった機能を活用することにより，不特定多数のユーザが利用する電子政府端末(図3)やガイダンス端末，街角に置くKIOSK端末などを容易に実現できる．

Teacubeの意義

Teacube/V$_R$5701評価キットは，PCなみの高い解像度とPMC T-Shellによる高機能なGUI，多漢字対応ブラウザである

注7：最大150万字の区別が可能なTRONコードの文字を，"&T～;"で囲んだ6桁の16進数で表現した形式．たとえば，「𠮷」のTRONコードは24D32，テキスト形式TRONコードは"&T224D32;"となる．シフトJISやUTFなどでエンコードされたテキスト・データ中にTRONコードの多漢字を埋め込むための手法として便利な方法である．

BBBなどを生かして，業務用端末，KIOSK端末，券売機，精算機などの用途に幅広く利用できる．一方，ブラウザや高機能GUIの実現といった点に注目すれば，PCを使って同じような機能を実現することも可能であろう．そう考えた場合に，PCとの比較におけるTeacubeのメリットとは何だろうか?

まず，Teacube/V$_R$5701評価キットの具体的なメリットについていえば，基板がコンパクトで各種の機械に組み込みやすいこと，ハードディスクのような機械的な可動部分がないため，振動に強く，工場や工事現場，車積向けの制御機器など環境の悪いところでも利用しやすいことなどが挙げられる．可動部分については，たとえばPCでもハードディスクの代わりにシリコン・ディスクを使えば，可動部分のないWindows PCを作ることは原理的に可能だろう．しかしながら，コスト面も含めて考えると，Gバイト近い大容量の2次記憶を必要とすることが多いWindowsベースのシステムをシリコン化するのは無理が大きい．それに対して，TRONのOSはコンパクトであり，多漢字のフォントを除けば，32Mバイト程度の安価なCFにWebブラウザを含むシステム全体が収まる．多漢字を含む17万字のフォントを入れたとしても，256MバイトのCFで運用できる．コスト面まで含めた現実性を考えながら可動部分のないシステムを作ろうとすれば，コンパクトであるというTRONのソフトウェアの特徴を生かすほうが良く，ハードウェアも含めてそのようなシステムを具体化したものがTeacubeなのである．さらに，Teacubeでは，PCよりも起動や終了の速度が速いといったメリットや，ハードウェア，ソフトウェアとも中身がわかっていてカスタマイズしやすいというメリットもある．

ところで，TeacubeとPCの比較といった話題を一般化していくと，T-Engineベースで開発された各種のユビキタス機器とPCとの比較論になるだろう．すなわち，組み込み機器やユビキタス機器とPCとの本質的な違いは何か，PCでユビキタス機器を代替できるのか，といった議論である．

PCの場合，CPUはx86系または互換CPUしか使えないし，周辺回路の主要部分も特定のチップセットによって実現されており，その内部はブラック・ボックスになっている．そのため，ハードウェアの自由度が少なく，いろいろな組み込み機器の要求に合わせてカスタマイズすることが難しい．たとえば，基板サイズをコンパクトにしたいとか，省電力化を進めて乾電池でも動くようなシステムにしたいといった要求には，簡単にはこたえられない．また，チップのコストが高いといった問題もあり，この点でも組み込み機器の要求に合わない場合が多い．

既存の組み込み機器を見ても，たとえばPCの何倍もの台数を出荷している携帯電話の場合，PCに匹敵する高機能なソフトウェアを搭載しているにもかかわらず，x86系のCPUを採用していないし，周辺アーキテクチャもPCとは異なっている．ゲーム・マシンについても同様の状況である．これは，それらの機器の用途に必要とされるハードウェアのコスト，サイズ，消費電力などの条件が，x86系のCPUとは折り合わず，ほかのCPUのほうが優れていたからにほかならない．

結局のところ，PCはx86系のCPUおよびWindowsという制約に縛られている限り，コスト，物理サイズ，消費電力などの条件が有利なわけではなく，組み込み機器での採用も限られている．これに対して，TeacubeのようなT-Engineベースのシステムであれば，ソフトウェアの開発効率をほとんど落とさずに，CPUやハードウェアの選択肢を広げていくことが可能であり，コストや消費電力などの点も含めて，組み込み機器のニーズに見合ったCPUを選べる．「Teacube/V$_R$5701評価キット」の場合，この製品はまだ評価キット(プロトタイプ)であって量産品ではないため，PCの量産品と比較して安価とはいえないものの，PCなみのロット数を量産すれば低コスト化が可能になるはずである．

Teacubeではさらに，高解像度の画面上で動くブラウザ(BBB)など，PCと同等の機能をx86系以外のCPUとWindows以外のOS上で実際に動かした点で，T-Engine以上の大きな意義がある．それぞれの組み込み機器で要求される仕様が「Teacube/V$_R$5701評価キット」やBBBと同一とは限らないが，要求に合わない部分はカスタマイズすればよい．まずは，実際に動くプロトタイプを短期間に開発し，最終製品の具体的なイメージを掴むことが重要である．Teacubeが，T-EngineベースのPCという意味で一つの実現例を示したことで，これまでPCを使っていた用途，たとえばインターネット端末のような用途にも，T-Engineベースのシステムが使えることを実証したのである．

組み込みシステムは，ハードウェアとソフトウェアを自分で開発すれば，原理的には何でもできる．しかし，最近の組み込みシステムはハードウェア，ソフトウェアとも高機能化，複雑化しており，すべてを自分で開発するのは非現実的になっている．現実的な開発期間や開発コストでプロジェクトを進めるには，開発の際の足がかり，すなわちプロトタイプになるようなコンピュータ・システム(ハードウェアとソフトウェア)が必要である．このときのプロトタイプとして有用なのが，小規模なGUIを使う用途であれば標準T-Engineであり，大画面の業務用端末的な用途であればTeacubeなのである．

参考文献，URL
(1) パーソナルメディアのT-Engineソリューション，T-Engine開発キット，Teacube，http://www.personal-media.co.jp/te/
(2) 京セラエルコのT-Engine/μT-Engine拡張バス専用コネクタ，http://www.kyocera-elco.com/guide/kiban/ml/sm/5603.html
(3) 佐々木 淳;「T-EngineにつなぐFPGAボード」，TRONWARE Vol.86, pp.50-55，パーソナルメディア，2004.

まつい・あきら　パーソナルメディア(株)開発本部

カーネルとアプリケーションをつなぐインフラ

第5章 T-Engineのミドルウェア

松為 彰

ハードウェアとOSは用意された．従来はこれだけでアプリケーションを開発することもあったが，現在の組み込み機器では，ファイル・システムやGUI，ネットワーク機能など，高度な機能が要求される．これらをゼロから作っていては，迅速な製品投入は難しい．そのため，これらの機能は「ミドルウェア」と呼ばれ，多数の実績ある実装が流通している．しかしμITRONでは，実行用ハードウェアやプログラムの作り方に関するガイドラインがないことから，ミドルウェアの流通性に問題があるという指摘があった．

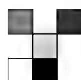
そこでT-Engineではこれらの問題を解決するために，統一的なプラットホームを規定し，ミドルウェアの流通を行いやすくした．本章ではこれらミドルウェアについて，実際に動かしながら理解する．（編集部）



はじめに

T-Engineや，その上で動作する標準リアルタイムOSのT-Kernelを使う最大のメリットは，ミドルウェアの互換性と流通性が高いことである．T-Engineプロジェクトでは，ミドルウェアの互換性を高めるために，開発評価用のハードウェアであるT-EngineボードやμT-Engineボードの仕様および機能を標準化し，ソフトウェアのモジュール化を強力にサポートする機能をT-Kernelに備えた（第3章p.61参照）．

ここではまず，こういったプラットホームの上で実際に開発/利用されているT-Engine用のミドルウェアに関して，概要および実装例を説明する．後半では，具体的なミドルウェアの例として，多漢字対応のGUIミドルウェア集であるPMC T-Shell(1)の機能や使い方，プログラミング例などを説明する．



T-Engineのミドルウェア

T-Engineプロジェクトが正式に発足してまだ2年だが，IPv6やVoIP，Bluetoothなどの通信およびネットワーク関連機能，MPEGやMP3，FLASH Playerなどのマルチメディア関連機能，指紋認証などのセキュリティ関連機能，GUI関連機能，多漢字関連機能，音声認識や音声合成，手書き文字認識など，すでに多くのジャンルのミドルウェアがT-Engine上に移植され，利用されている．

また，ブラウザなどの大型アプリケーションや，各種のデバイスに対するデバイス・ドライバも，数多く用意されている．この中には，ITRONやBTRONなど，従来のTRONの成果をT-Kernel上に移植したものも含まれている．T-Engineのミドルウェアを短期間に充実させることができたのは，T-Engine自体に対する注目度や期待が高く，開発リソースを集中できたということももちろんあるが，TRONプロジェクトの既存の資産を活用できたという理由も大きい．デバイス・ドライバに関しても同じことがいえる．

これらのミドルウェアは，TRONSHOW(2)や組み込みシステム関連の展示会，T-Engineフォーラムの会員向け説明会などで随時紹介されているほか，T-Engineフォーラムの中でT-Engine用ミドルウェアの登録制度を設け，ミドルウェアの情報を体系的に提供できるような体制ができつつある．また，T-Engine開発キット上ですぐに動かせる開発評価用のミドルウェアがオブジェクト・コードで比較的安価に販売されている場合もあり，PCにアプリケーションをインストールするのと同様の手軽さでこれらのミドルウェアを試用したり，これらのミドルウェアを使った最終製品のプロトタイプを短期間に開発することもできる．

ちなみに，従来の組み込み機器用のミドルウェアでは，実行環境を固定できなかったためにオブジェクト・コードでの供給が難しく，ユーザ自身が自分の環境に合わせてソース・プログラムを移植する必要があった．この手法ではユーザ側にとって手間や時間がかかるほか，ミドルウェアの開発者側にとってもソース・プログラムでの供給が必須となるため，価格やライセンス体系に制約を受ける．T-Engineの利用により，こういった制約が解消され，ミドルウェアの開発評価版を安価なオブジェクト・コードで供給できるようになった．すなわち，ビジネス形態の自由度が拡大したわけである．こういった点からも，T-Engineプロジェクトが組み込み業界の活性化に寄与する点は大きい．

以下，T-Engineの代表的なミドルウェアやアプリケーションのうち，開発評価版を購入できるものや，TRONSHOWなどの展示会でデモンストレーションが行われたものを中心に，概況を説明する．

● ファイル管理機能と開発用の基本ミドルウェア

ミドルウェアのうちでも汎用性の高いファイル管理機能については，T-Kernel Standard Extensionの一部として，T-Engine開発キットに付属して提供される（MMU非対応の一部

のμT-Engineを除く）．T-Kernel Standard Extensionのファイル管理機能は，ハイパーテキストやマルチレコードに対応したTRON形式のファイル・システムをサポートする．また，UNIXエミュレータとの組み合わせにより，FAT形式のファイル・システムやCD-ROM用のISO 9660形式のファイル・システムを扱うことも可能であり，WindowsなどのPCはもちろん，ディジタル・カメラなどとのデータ交換も容易である．T-Kernel Standard Extensionのファイル管理機能は，開発，デバッグ時のプログラムのダウンロードにも利用される．

T-Engine開発キットには，USBバス・ドライバやPCMCIAバス・ドライバ，キーボードやマウスのドライバ，USB Mass Storage Classのドライバなど，T-Engineボードの周辺I/Oに対するデバイス・ドライバも付属している．これらのデバイス・ドライバとファイル管理機能との組み合わせにより，PCMCIAスロット経由でCFカードにアクセスしたり，USB経由でハードディスクやCD-ROMドライブ，一部のディジタル・カメラなどにアクセスし，その中のファイル（TRON形式，FAT形式など）を操作することが可能である．

ちなみに，T-Kernel Standard Extensionは，PC用のTRONのOSとして設計されたBTRONの周辺核（ファイルなどを扱う階層）をベースに，T-Engine向けの修正や機能追加を行ったものである．T-Kernel Standard Extensionには，ファイル管理以外にも，プロセス管理，イベント管理，仮想記憶対応機能など，プログラムのモジュール化や開発効率の向上に役立つ便利な機能が含まれている．このため，T-Kernel Standard ExtensionはT-Kernel上で動くミドルウェアであるにもかかわらず，T-Engineのシステム全体から見るとOSの一部のような存在になっている．

T-Engineのミドルウェアやアプリケーションは，直接T-Kernelの上で動くものと，T-Kernel Standard Extension（というミドルウェア）の上で動くものに分類される．前者はT-Kernelベースのプログラムと呼ばれており，メモリ空間が分離されず，プログラム全体を一つにリンクしていた従来のITRONアプリケーションに近い性質を持っている．一方，後者はプロセス・ベースのプログラムと呼ばれている．アプリケーション（プロセス）ごとにメモリ空間が分離され，動的なローディングや実行の環境を積極的にサポートし，UNIXやPCのアプリケーションに近い性質を持つのがプロセス・ベースのプログラムである．動的なローディングに関しては第3章を参照していただきたい．

● TCP/IP

T-Engine用のTCP/IPは，すでに多くの実装例が発表されているが，そのうち代表的なものの一つが（株）エルミックシステムの開発した「KASAGO for T-Engine」である．KASAGOは組み込みシステム向けのミドルウェアとして高速性や信頼性に優れ，カスタマイズの柔軟性が高いほか，IPv6にも対応したデュアルスタック構造を持つのが大きな特徴である．また，「T-Engine/SH7727開発キット」，「μT-Engine/M32104開発キット」にインストールしてすぐに使える「KASAGO for T-Engineバイナリ評価ライセンス」が販売されており，試用や評価が容易である．

一方，後述する「PMC T-Shell開発キット」や第4章の「Teacube/V$_R$5701評価キット」には，KASAGOとは別の実装によるTCP/IPが含まれている．こちらは，IPv6には対応していないが，KASAGOとは別の機種もカバーしており，用途に応じて使い分けることができる．

● GUI関係のミドルウェア

GUI関連のミドルウェアにも多くの実装例があるが，代表的なものの一つは，メトロワークス（株）が開発したGUIツールキット「PowerParts」である．PowerPartsは，組み込みシステム用のGUIフレームワークと，レイアウト用のRAD（Rapid Application Design）ツールから構成されており，グラフィック画面を備えた組み込み機器のGUIを，効率よくビジュアルに開発できる．T-Engine用のPowerPartsを使ったブラウザなどの実装例が，すでにTRONSHOWなどで展示されている．

「PMC T-Shell開発キット」や「Teacube/V$_R$5701評価キット」にも，GUI関連のミドルウェアが含まれている．こちらは，PC用に設計されたBTRONの外殻（ウィンドウなどを扱う階層）をベースに，T-Engine向けの修正や機能追加を行ったものである．PMC T-Shellについては後述する．

● マルチメディア関係のミドルウェア

JPEG，MPEG，MP3など，多くのミドルウェアがT-Engine上に移植されており，それぞれに複数の実装例がある．JPEGは，PMC T-Shellに含まれているほか，別の実装のものが（株）日立超LSIシステムズのミドルウェア集である「HI ApplicationEngine for T-Engineミドルウェアセット」に含まれている．このミドルウェアセットは，SH7727版やSH7751R版のT-Engineで動作し，音声合成やMP3の機能も提供している．日立超LSIシステムズは，T-Engineの応用製品であるユビキタスコミュニケータ（UC）上にもMPEGなどのミドルウェアを供給している．

サウンドや音声関係のミドルウェアとしては，上記のMP3のほか，携帯電話の着信メロディの市場で高いシェアを持つ（株）フェイスが，自社のソフトウェア・シンセサイザ（カラオケ・アプリケーション），音声デコーダ，音声エフェクタ，音声読み上げエンジンなどをT-Engine上に移植している．また，NECエレクトロニクスでは，T-Engine/V$_R$5500の上で動作する音声合成および音声認識のミドルウェアを開発している．

T-Engine上に移植されたモバイル向けの3Dポリゴン・エンジンとして，（株）エイチアイの開発した「Mascot Capsule for T-Engine」がある．リソースの厳しい携帯機器や家電製品の画面上で，3次元のグラフィックを生かした豊かな表現力を実現できるミドルウェアであり，このような製品のプロトタイプをT-Engine上で開発できる．

昨年末のTRONSHOWでは，東芝情報システム(株)がMacromedia社のFLASH PlayerをT-Engine上に移植し，デモンストレーションを行った．移植先となった機種は，東芝製のCPUを搭載したT-Engine/TX4956である．最近では，街角に置かれる自動販売機などにも小型のディスプレイが搭載され，ちょっとした広告などのアニメーションが流れる例が増えている．FLASH Playerが動けば，こういった用途にもT-Engineを採用しやすくなり，T-Engineの応用製品のさらなる拡大が期待できる．

● セキュリティ関係のミドルウェア

昨今の組み込み機器の制御用ソフトウェアは，機能や使いやすさだけではなく，不正利用やシステムのクラッキングに対する防御機能がないと，実用的なシステムとはなり得ない．このような状況においては，暗号化や生体認証など，セキュリティ関連の機能も重要である．

TRONプロジェクトでは，ICチップやICカードを使った認証システムにも応用できるセキュリティ・アーキテクチャとして，eTRONを開発している．T-EngineやTeacubeでは，eTRONチップの挿入用スロットがついており，eTRONを応用したシステムの開発も可能である．

また，パーソナルメディア(株)では，一部のT-Engine開発キットやTeacube/V_R5701評価キット上でeTRONカードの応用システムを開発できる「eTRON/8開発キット」を販売している．本キットには，非接触タイプのeTRON/8カード，T-EngineやTeacubeのUSB端子に接続可能なeTRONカードのリーダ/ライタ，eTRON/8カードのアクセス用ライブラリ，ドキュメントなどが含まれており，簡単な手順でeTRONカードを試用したり，電子マネー，電子チケット，Eコマース関連の応用システムのプロトタイプを開発することが可能である．

セキュリティ関連では，このほか，日立エンジニアリング(株)の開発したT-Engine用の指紋認証システム「Finger Attestor for T-Engine」がある．本製品には，指紋認証用のミドルウェア(プログラム)のほかに，静電容量方式のスイープ型指紋センサを搭載したT-Engine用の拡張ボードが含まれており，T-Engine/SH7727開発キットに接続して利用できる．ちなみに，「スイープ型のセンサ」とは，ぺったりと指を置くタイプの面型のセンサではなく，指を滑らせるタイプのコンパクトなセンサであり，携帯電話やPDAなどのモバイル機器，小型の組み込み機器にも搭載しやすいという特徴がある．

● 大型ミドルウェアと大型アプリケーション

一般的な意味でのミドルウェアの範ちゅうには含まれないかもしれないが，もともとはTRONと別だったOSやソフトウェア体系をT-Kernel上に移植し，双方のメリットを生かせるハイブリッドなシステム構成とした例も見られる．代表的なものは，T-EngineやT-Kernel上にLinuxを載せたT-Linuxと，Javaを載せたT-Javaである．

この場合のLinuxやJavaは，T-Kernel上で動く汎用的なソフトウェアであり，さらにその上で個々の最終製品に依存した制御用のアプリケーションが動くという意味で，ミドルウェアと同じ位置付けを持つ．昨年の発表で大きな話題になったWindows CE.NETも同様であろう．さらに，T-Engine用のGUIミドルウェア集であるPMC T-Shellも，BTRONという大きなOSのウィンドウシステムの階層をT-Kernel上に移植したものという意味で，似たような位置付けを持つ(図1)．

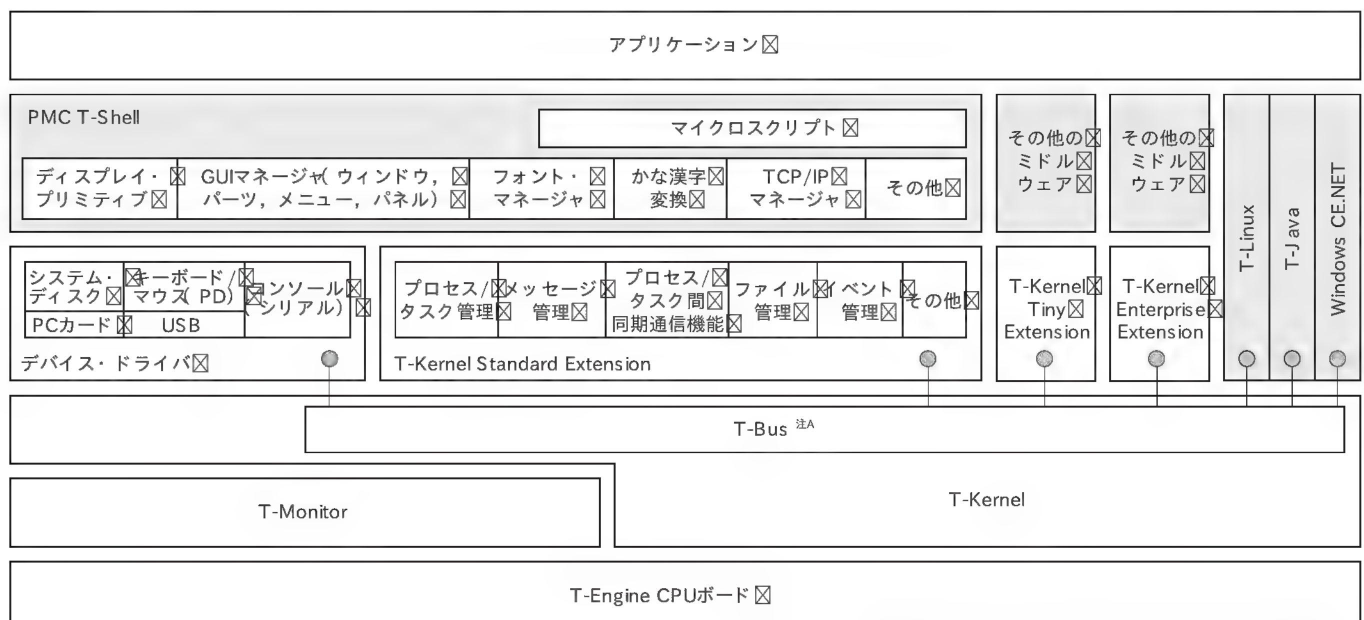


図1　T-Engineのソフトウェア構成図



図2　PMC T-Shellを利用した電子ブックの画面例

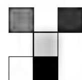


図3　PMC T-Shellを用いた人名用多漢字の検索と活用例

T-Kernel上で動作するWindows CE.NETとT-Linuxに関しては，第6章と第7章を参照してほしい．

一方，ミドルウェアに限らず，汎用的なアプリケーションについても，ブラウザ類を中心に多くの実装例，移植例がある．Webブラウザについては，パーソナルメディアがBBB(BTRON Basic Browser)をT-EnigneやTeacube上に移植しているほか，Mozilla Firebird(新名称 Mozilla Firefox)や，ウェブソフト・インターナショナル(株)の組み込み向けブラウザである「Esprit(エスプリ)」などの実装例がある．

このほか，(株)KDDI研究所は，地図などの表示に適したSVG(Scalable Vector Graphics)のブラウザをT-Engine用に開発している．また，ピクセル・テクノロジーズ(株)は，Word，Excel，PowerPointなどのMS OfficeファイルやPDFファイルの閲覧が可能な「Picsel ePAGE」というブラウザをT-Engine用に移植している．さらに，日本オラクル(株)は，「Oracle9i Lite for T-Engine」の名称で，Oracleデータベースの軽量版をT-Engine上に移植している．



ミドルウェアPMC T-Shell

T-Engineで動くミドルウェアの具体例の一つとして，パーソナルメディアの多漢字GUIミドルウェア集である「PMC T-Shell」について解説する．ちなみに，PMC T-Shellの“Shell”とは貝殻の「殻」を意味する語であり，PMC T-Shellのベースとなっている BTRONのウィンドウ・システムの階層(外殻部分)を英語で“Shell”と呼んだことに由来している．すなわち，OSのカーネル(核)に相当するT-KernelやT-Kernel Extensionに対して，その外側(上位側，アプリケーション側)のミドルウェアであるという位置付けを表すのが“Shell”の意味である．なお，“PMC”とは開発元のパーソナルメディア社の略称である．

PMC T-Shellには，グラフィック画面制御などのGUI機能と，多漢字機能[注1]，ビジュアル言語「マイクロスクリプト」の実行環境，ネットワーク接続用のTCP/IP機能など，多くの有用な機能が含まれている．T-Kernel ExtensionにPMC T-Shellを組み合わせて利用することにより，ソフトウェア環境の充実したPCとほぼ同様の実行環境をT-Engine上にも整えることができ，AV機器，OA機器，券売機，KIOSK端末など高機能なGUIを持つ組み込み機器を容易に開発できる．さらに，多漢字の需要が大きな電子ブック(図2)や電子辞書，電子政府向け端末など，文字を扱う組み込み用途にも，幅広い応用が可能である(図3)．

PMC T-Shellは，SH版，V_R版，ARM版などほとんどの標準T-Engineのほか，T-Engineベースの応用製品であるユビキタスコミュニケータやTeacube/V_R5701評価キットにも移植済みであり，これらのCPUを用いたユビキタス機器の開発効率向上に貢献している．

また，PMC T-Shellの上で動くアプリケーションはもちろん，PMC T-Shell自身も再コンパイル程度の手間で容易にほかのCPUを搭載したT-Engineボードに移植することができ，ミドルウェアの流通基盤を確立するというT-Engineプロジェクトの狙いを実証する役割を果たした．それに加えて，PMC T-Shellの開発の経緯から当然のことではあるが，PC上で動くBTRON仕様OS「超漢字」のGUI関連機能も，PMC T-Shellと高い互換性を持っている．このため，PMC T-Shellで動くアプリケーションのうち，ハードウェアへの依存性が少ない部分(GUI画面の制御部分やファイル関連，LAN関連の部分など)については，PC上で先行して開発作業を進めておくことができる．すなわち，PC，T-Engine，最終製品といった実行環境の違いをあまり意識することなく，シームレスに開発やデバッグ作業を進めることができ(図4)，ソフトウェアの開発効率を大幅に向上できる．

注1：従来のコンピュータで扱える漢字(たとえばJISやUnicodeの文字)と比較して，より多くの種類の漢字を扱える機能を「多漢字機能」と呼ぶ．多漢字機能は，人名用の漢字を正確に扱う際に重要な機能である．



図4　PMC T-Shellを用いたシームレス開発のイメージ



図5　PMC T-ShellのGUIマネージャにより提供される機能

JIS第1・第2水準	亜弌漢あ。
JIS第3・第4水準	
JIS補助漢字	
GT書体フォント	
大漢和辞典	
韓国漢字・ハングル	
中国簡体字	
中国伝統字	
iモード絵文字	
点字	
記号その他	
Unicode	

図6
PMC T-Shellの標準パッケージで利用可能な文字

以下，PMC T-Shellの持つ機能と特徴的な点を説明する．

● ディスプレイ・プリミティブ

直線，円などの図形やビットマップ(画像データ)，文字，文字列などを画面上に描画する機能である．複数の描画環境(描画のコンテキスト)を切り替えることにより，一つの画面上に複数のプログラムから並行して描画できる．また，実際には描画を行わない領域(クリッピング領域)を設定することも可能である．

これらの機能は，PMC T-ShellのGUIマネージャにおいて，画面上に複数のウィンドウを重ねて表示するオーバラッピング・マルチウィンドウの機能を実現するために利用される．

● 画像形式変換ライブラリ

JPEG，PNG，BMP形式の画像データと，TRONのオリジナル形式(TAD：TRON Application Databus)との相互変換機能である．この機能とディスプレイ・プリミティブ，およびT-Kernel Extensionのファイル管理機能を利用することにより，ディジタル・カメラで撮影した写真の画像データ(画像形式はJPEG，ファイル形式はFAT)をT-Engineの画面上に表示するようなプログラムも容易に記述できる．ディジタル・カメラなど，画像を扱う組み込み機器のプロトタイプを開発する際には，極めて有用な機能である．

● GUIマネージャ

GUIマネージャは，PC用のOSにおけるウィンドウ・システムに相当する機能を提供する．具体的には，画面上のGUIスイッチ類やボリューム(スクロール・バー)などを管理するパーツ管理機能，ポップアップ・メニューを管理するメニュー管理機能，対話的なユーザ設定などをサポートするパネル管理機能，マルチウィンドウの管理を行うウィンドウ管理機能などが含まれる．

PMC T-ShellのGUIマネージャにより提供される機能の画面例を図5に示す．

● 多漢字機能とフォント・マネージャ

JISやUnicodeで定義された文字を含めて最大150万字を利用可能な多漢字機能と，それをサポートするフォント管理機能を提供する．PMC T-Shellの標準パッケージ(PMC T-Shell開発キット)で利用可能な文字は，現在約17万字あり(図6)，電子政府向けの人名用漢字を中心に幅広い漢字の需要を満たしている．また，必要に応じて，コンピュータ・メーカの標準外字セットや他の文字セット(住基ネット用統一文字など)，ユーザの持つ既存の外字セットなどを追加して利用することも可能である[3]．

多漢字を含む文字のフォントには，ドット形式とアウトライン形式(TrueType形式)の両方が利用可能であり，指定された文字サイズに応じて最適な形式のフォントが自動的に選択されて表示される．また，明朝やゴシックなどの書体を指定したり，指定された書体の文字フォントが存在しない場合に，それに近い書体の文字を自動的に選択して表示する機能もある．フォント・マネージャは，ディスプレイ・プリミティブの文字

描画機能から呼び出される形で，これらの処理を行う．

● かな漢字変換

日本語入力のために必要な，かな漢字変換の機能を提供する．PMC T-Shellに標準添付されているかな漢字変換機能は，バックス社の開発したVJE-DeltaをT-Engine上に移植したもので，連文節変換，学習機能，ユーザ辞書登録，多漢字機能などを備える．たとえば,「髙」,「﨑」,「𠮷」などの多漢字を含む人名と，それに対する読み方をユーザ辞書に登録しておけば，これらの文字をかな漢字変換で入力することが可能となる．

● TCP/IPマネージャとLANドライバ

TCP/IPのプロトコル・スタック（ICMP，ARP，DNS，DHCP，PPPのクライアント側の機能を含む）と，その下で動くLANドライバを提供する．LANドライバは，T-Engine用の拡張LANボード（SH7727版などの場合）のほか，標準T-EngineのPCMCIAスロットに挿入されるLANカードの一部にも対応している．

なお，T-Engineボードの標準ハードウェア仕様にはLANの機能が含まれておらず，オプションとなっていたため，それに対するドライバやTCP/IPもT-Engine開発キットには含まれていない．TCP/IPやLANドライバが別売オプションのソフトウェアとして供給され，GUIとは直接関係しない機能であるにもかかわらずPMC T-Shellに含まれているのは，こういった経緯による．

● マイクロスクリプト

「マイクロスクリプト」とは，以前から超漢字などBTRON上で動作していたビジュアル指向言語である(4)(5)．PMC T-Shellの中にもこの言語の処理系が含まれており，HMI（Human Machine Interface）のプロトタイプ開発（図7）はもちろん，ゲーム，シミュレーション，電子ブック，スライド・ショー，デモ・プログラムなど，多くの用途に活用できる．

マイクロスクリプトでは，画面に表示したい図形や画像データ（写真も可能）などの表示要素（セグメント）に名前をつけておき，それらのセグメントを，プログラムで指定した位置に表示したり，移動したりすることができる．また，セグメント上で起こったイベント（マウスのクリック操作など）に対するアクション（動作）を，平易なプログラムで記述できる．マイクロスクリプトを利用すれば，OSであるT-KernelやT-Kernel Extension，C言語などの知識がなくてもプログラムを開発できるだけでなく，C言語によるプログラム開発と比較してプログラムの修正やデバッグが容易というメリットもある．

マイクロスクリプトでは，実行時にCPUから独立した中間コード（仮想マシン・コード）に落としてから実行するというコンパイラ・インタプリタ方式を採用している．そのため，マイクロスクリプトのプログラムはCPUに依存せず，別のCPUを搭載したT-Engineや超漢字の動いているPC上でも，まったく同じプログラムを実行できる（図4）．T-Engineのミドルウェアやアプリケーションは，一般には再コンパイルするだけで別の機種に移植できるが，マイクロスクリプトで書かれたアプリケーションに関しては，再コンパイルさえも必要ないのである．したがって，ターゲット側のハードウェアが動作する以前から，PCと超漢字を用いてHMI関連のプログラムの開発やテスト，評価を進めておくことができ，開発期間の短縮や品質の向上に役立つ．また，画面の大きなPCを利用して，効率よくHMI画面の設計や開発を行えるというメリットもある．

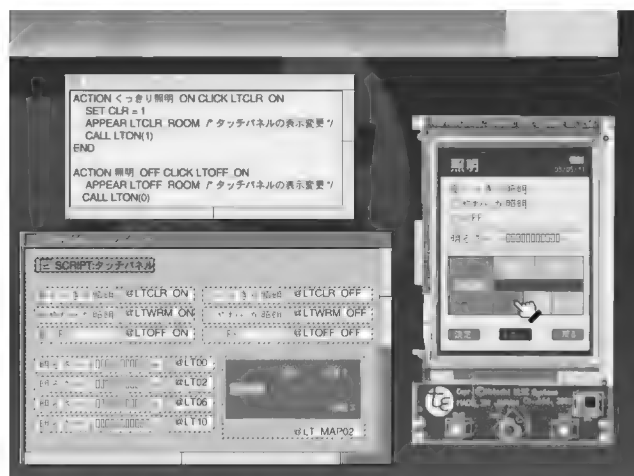


図7　マイクロスクリプトにより記述した照明制御用のGUI画面

マイクロスクリプトによる具体的な開発イメージは，先ほどの図4のようになる．図4では，パソコン上に表示されたT-EngineのLCD画面（画面中央）の中で，T-Engine用に開発したパソコン上のソフトウェア（画面左側：ボールのシミュレーション）と同じものが動く．さらに，T-Engineベースの応用製品であるユビキタスコミュニケータ上でも，同じソフトウェアが動く（画面右側）．C言語で書かれたアプリケーションは再コンパイルで，マイクロスクリプトで書かれたアプリケーションはまったく同じプログラムがそのまま動く．

PMC T-Shellのプログラミング例

以下に，PMC T-Shellを使ったサンプル・プログラムと，その実行例を示す．これらのサンプル・プログラムのソースの全文は，本誌付属CD-ROMに収録されているので，合わせてご覧いただきたい．このほか，参考文献の（6），（7），（8）にも，これらのサンプルに対する解説記事が掲載されている．

● ディスプレイ・プリミティブ編

まず，PMC T-Shellのディスプレイ・プリミティブを使って，T-Engineの画面に簡単な図形を描くC言語プログラムの例（`tshell_1.c`）を示す．`tshell_1.c`のソースの主要部をリスト1に，T-Engine上での実行結果を図8に示す．また，`tshell_1.c`で利用したT-Shellの機能（API）を表1に示す．このサンプルは，画面の適当な場所に四角形の枠を描くだけの

リスト 1　図形描画プログラム(tshell_1.c)

```
/*
 *  tshell_1.c (T-Shell sample/メイン)
 *  (C) Copyright 2004 by Personal Media Corporation.
 */
#include    <basic.h>
#include    <btron/outer.h>
#include    <btron/dp.h>
#include    <tcode.h>
#include    <stdio.h>
#include    <stdlib.h>

/* --------------------------------- main(cli style main) */
EXPORT  W   main(W ac, TC *av[])
{
        ...

    /* 5. デバイス描画環境を生成する */
    gid = b_gopn_dev(SCREEN_NM, NULL);
    printf("b_gopn_dev : %d, %d¥n", gid, gid >> 16);
    if (gid < 0) {
        /* 失敗 */
        rv = gid;
    } else {
        /* 6. 背景として全面を塗りつぶす */
        r = (RECT){{0, 0, scr_spec.hpixels,
                       scr_spec.vpixels}};
        b_gfil_rec(gid, r, BLACK100, 0, G_STORE);

        /* 描画 */
        for (l = 0; l < 10; l++) {
            /* 7. 枠の座標を生成 */
            r.c.left = (rand() * scr_spec.hpixels) / RAND_MAX;
            r.c.top = (rand() * scr_spec.vpixels) / RAND_MAX;
            r.c.right = (rand() * scr_spec.hpixels) /RAND_MAX;
            r.c.bottom = (rand() * scr_spec.vpixels)
                                                / RAND_MAX;
            if (r.c.left >= r.c.right) {
                i = r.c.left;
                r.c.left = r.c.right;
                r.c.right = i;
            }
            if (r.c.top >= r.c.bottom) {
                i = r.c.top;
                r.c.top = r.c.bottom;
                r.c.bottom = i;
            }

            /* 8. 枠を描画する */
            b_gfra_rec(gid, r, 0x0001, WHITE0, 0, G_STORE);
            printf("r : %d, %d, %d, %d¥n",
                   r.c.left, r.c.top, r.c.right, r.c.bottom);
        }

        /* 9. 描画環境を削除する */
        b_gcls_env(gid);
    }

    /* 10. プロセスを終了する */
EXIT:
    b_ext_prc(rv);
    return rv;
}
```

簡単なものであり，基本的な処理の流れは次のとおりである．

a) 描画対象デバイスの確認と描画環境の生成

b_gget_spcにより対象デバイスの情報(VRAMの解像度や属性など)を取得した後，b_gopn_devによってデバイス描画環境を生成し，描画の準備を行う．

b) 乱数による座標値の取得と四角形の描画

背景となる画面全体を黒で塗りつぶした後，forループの中でb_gfra_recを実行し，画面上に多数の四角形(の枠)を描画する．四角形の座標値は，乱数によって取得する．

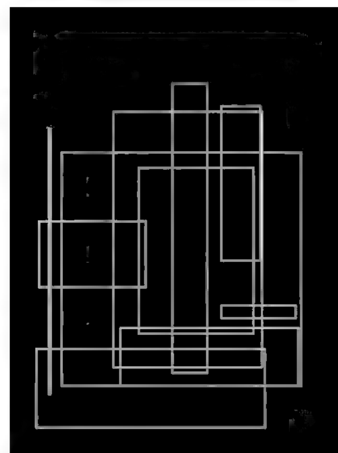


図 8　図形描画プログラム(tshell_1.c)の実行例



図 9　文字および多漢字の表示例(tshell_2.c)

表 1　tshell_1.cで利用したT-Shellの機能

b_gget_spc()	デバイス情報の取り出し
b_gopn_dev()	デバイス描画環境の生成
b_gcls_env()	描画環境の削除
b_gfil_rec()	長方形の塗りつぶし
b_gfra_rec()	長方形の枠の描画

c) 終了

b_gcls_envにより描画環境を削除した後，b_ext_prcによりこのプロセスを終了する．なお，b_ext_prcはT-Kernel Extensionの機能である．

● 多漢字の表示例

前項のtshell_1.cをベースに，四角形ではなく文字を描画するように修正したサンプルがtshell_2.c(CD-ROM参照)である．tshell_2.cの実行例を図9に示す．この中では，JIS範囲内の文字のほか，多漢字としてGT書体フォントも表示している．

● JPEG画像ビューア

tshell_3.c(CD-ROM参照)は，ディジタル・カメラなどで撮影したJPEGの画像をT-Engineの画面に表示するJPEG

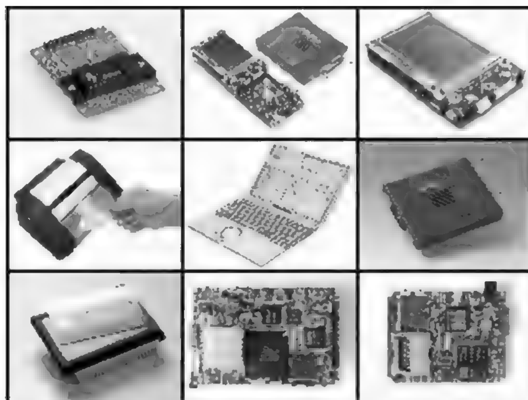

図10 JPEG画像ビューア(tshell_3.c)の実行例
一覧モードで9枚の写真を表示している



図11 マイクロスクリプトのサンプル(tshell_4)の図形部分

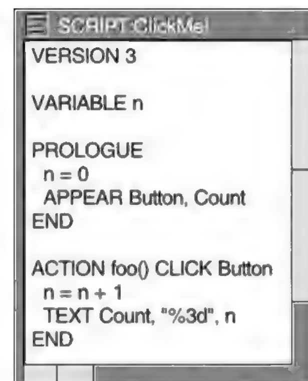


図12 マイクロスクリプトのサンプル(tshell_4)のスクリプト部分



図13 マイクロスクリプトのサンプル

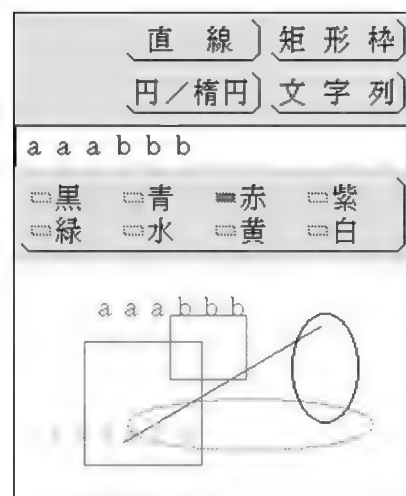


図14 C言語とマイクロスクリプトの連携プログラム(tshell_5)の実行例

画像ビューアのサンプル・プログラムである．撮影した画像はCFやSDカード内のFAT形式のファイルとして記録されているので，このサンプルでは，まずT-Kernel ExtensionのFATアクセス機能を利用して，画像ファイルを読み出す．なお，FATアクセス機能はUNIXエミュレータというサブシステムに含まれているので，このサンプルを実行する前に，lodspg unixemuというコマンドでこのサブシステムをロードしておく(設定により自動化も可能)．また，標準T-EngineにはSDカードのスロットが付いていないため，SDカードを読むにはUSB接続のSDカード・リーダを利用する．

このようにして読み込んだ撮影画像のファイルは，JPEG形式になっているので，PMC T-Shellの画像形式変換ライブラリを使ってTRONの画像形式に変換した後，ディスプレイ・プリミティブの機能を使ってT-EngineのLCDに表示する．

さらに応用的な機能として，画面を9分割して複数の写真を同時に表示する一覧モードの機能(図10)や，一覧モード時にタッチ・パネルで選択した写真を拡大して表示する機能，LCDボードに付いているボタンによって表示する写真を選択する機能などを備えている．

● マイクロスクリプト編

次に，T-Shellのマイクロスクリプトで動く簡単なサンプルプログラム(tshell_4)を示す．マイクロスクリプトのアプリケーションは，画面に表示するセグメントの集合体である図形部分(図11)と，動作シーケンスを記述するスクリプト部分(図12)からなっている．このプログラムを実行すると，画面上に"Click me!"と書かれたスイッチが表示され，その部分をクリックすると，スイッチの上に表示されている数字が一つ大きくなる(図13)．マイクロスクリプトを使うと，こういったGUI関連の処理を簡単に記述できるほか，プログラマの作るスクリプト部分とデザイナの作る画面デザインを独立して開発できるというメリットがある．

● C言語とマイクロスクリプトの連携

最後に，GUI画面の構築が容易というマイクロスクリプトのメリットと，処理の自由度が高いC言語のメリットを組み合わせたサンプルをtshell_5(CD-ROM参照)に示す．このプログラムを実行すると，図14のような画面が表示される．この状態で，たとえば「直線」のボタンをクリックした後に画面下部の2点をクリックすると，その2点を結ぶ直線が描画される．「矩形枠」，「円/楕円」についても同様である．また，「文字列」をクリックすると，画面上部の枠内に入力された文字列を画面下部に表示する．表示の際，色を選ぶことも可能である．

マイクロスクリプトとC言語の連携にはいくつかの手法があるが，この例では，プロセス間メッセージの機能を利用している．具体的には，マイクロスクリプトのMSEND文により送信されたメッセージをC言語のAPIのb_rcv_msgで受信でき，C言語のb_snd_msgにより送信されたメッセージをマイ

クロスクリプトのMRECV文によって受信できるので，この方法により両者のプログラム間でメッセージ通信を行い，相互に連携した処理を実現する．tshell_5の場合，直線や円を描画する部分はC言語によって処理され，「直線」，「矩形枠」，「円/楕円」，「文字列」のボタンや色指定のスイッチなど，GUIに関する部分はマイクロスクリプトで処理されている．

おわりに

T-Engineフォーラムでは，T-Engine用のミドルウェアに対する新たな試みとして，eTRONを用いたミドルウェアの課金システムを構築する予定である．これは，T-Engine，μT-EngineのボードやTeacubeに付属しているeTRONの挿入用スロットに，課金情報（ライセンス・チケット）を保持したeTRONチップを挿入し，そこからミドルウェアの使用料金を徴収するというメカニズムである．eTRONによる電子的な課金方法の確立により，ミドルウェアのプログラム自体はオンラインで自由に配布できるようになる．また，課金方法についても，初回にいくらとか，実行回数や利用期間に応じていくらとか，マーケティングの要求に応じたいろいろな課金体系が可能となる．このシステムは「T-Dist」と呼ばれており，まもなく実験が始まる予定である．T-Distでは，まずT-Engine開発キット上で実行される開発評価用のミドルウェアを対象として実験や運用を開始するが，将来的には，eTRONスロットを備えたT-Engineベースの最終製品や，プログラム以外のデータウェア，コンテンツなどにも応用が拡がっていくと期待される．

T-Engine用のミドルウェアは，今年から来年にかけて，質，量ともにますます充実し，ユビキタス機器の開発を支えるソフトウェアの大黒柱として，T-Engineの普及に貢献していくことであろう．

参考文献，参考URL

(1) パーソナルメディアのT-Engineソリューション，PMC T-Shell，http://www.personal-media.co.jp/te/
(2)「ユビキタス，TRONに出会う ～TRONSHOW 2004レポート～：出展者ブース概要」，TRONWARE VOL.85 pp.36-42，パーソナルメディア，2004.
(3) 松為 彰；「多漢字問題とTRONコード」，Interface，2002年12月号，pp.75-86，CQ出版社，2002.
(4) 坂村 健 監修/PMC研究所 編；「マイクロスクリプト入門」，パーソナルメディア，1999.（ISBN4-89362-160-2）
(5) 坂村 健 監修/PMC研究所 編；「BTRONマイクロスクリプト」，パーソナルメディア，1997.（ISBN4-89362-155-6）
(6)「PMC T-Shellの実践的活用法 ～総論およびディスプレイプリミティブ編～」，TRONWARE VOL.82，pp.49-58，パーソナルメディア，2003.
(7)「PMC T-Shellの実践的活用法 ～マイクロスクリプトとC言語よるプログラムの連携編～」，TRONWARE VOL.83，pp.32-40，パーソナルメディア，2003.
(8)「T-EngineからFATファイルシステムをアクセスする」，TRONWARE VOL.84，pp.94-97，パーソナルメディア，2003.

まつい・あきら　パーソナルメディア（株）開発本部

豊富な資産を継承するための

第6章

Windows CEとT-Kernelの協調動作の原理

芝本 利博/岸 恵一/小田桐 康弘

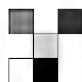
昨年発表されたWindows CEとT-Kernelを協調動作させる試みは大きな話題を呼んだ．TRONSHOW 2004で発表された実装例は，T-Engine上でWindows Media Playerが動作するというインパクトのあるものだった．

これまでも複数のOSを同時に動作させる試みはあったが，Windows CEとT-Kernelの場合はそれらとは違い，間にブリッジ・モジュールを介在させ，それらの間で通信を行うことにより協調動作を行わせる「ブリッジ・フレームワーク」を採用するという点が技術的に目新しい．

そこで本章では，このブリッジ・フレームワークを中心に，Windows CEとT-Kernelを協調動作させる原理について解説する．（編集部）

はじめに

昨年9月25日にMicrosoft社とT-Engine Forumは次世代プラットホームの実現に向けて協力すると発表しました．今回はその協力の中で検討してきた一つの実例であるT-Engine上でのWindows CEとT-Kernelの協調動作の枠組みを解説します．



改めて解説するWindows CE

● Windows CEとは

Windows CEは，リアルタイムOS本体と，統合開発環境“Platform Builder”の二つにより構成されています．豊富なミドルウェア，アプリケーション，プロトコル・スタックを備え，広範囲なCPUおよびデバイスをサポートすることによって，Windows CEは次世代のネットワーク対応スマート・デバイスを少ない追加コストで構築することができます．Platform BuilderはGUIベースの開発環境であり，カーネル・デバッガ，ハードウェア・デバッグ・インターフェース（eXDI），テスト・ツール，エミュレータ環境などを備えています．

また，Microsoft eMbedded Visual C++による従来のネイティブ・アプリケーション開発のほか，Microsoft Visual Studio .NETによるWebサービスやプラットホームに依存しないアプリケーションも開発可能です．

本節では，Windows CEの機能やテクノロジについて解説します．

● Windows CE製品のロードマップ

Windows CEは，携帯端末向けのベースOSとして開発が開始されました．2000年にリリースされたバージョン3.0では，組み込み機器向けにリアルタイム性の改良が行われ，その後，現在に至るまで，さまざまな機能追加が行われています（図1）．Microsoft社製品の中では，Windows CEはPocket PC/Smartphone（携帯電話）のほか，Windows Automotive（カー・ナビゲーション）のベースOSとして使用されています．

● Windows CEの特徴

Windows CEは，ROMやRAMに制約のある組み込み機器向けのコンパクトな実装を可能としたリアルタイムOSですが，同時に以下のような多くの特徴を備えています．

1）同梱される豊富な機能

Windows CEには，最新のネットワーク機能，マルチメディア機能，デバイス・ドライバなどがバイナリまたはサンプル・ソースとして同梱されており，そのまま，あるいはカスタマイズして必要な機能だけを取り出して，製品に利用できます．

以下，Windows CEに含まれる機能の一例を示します．

- Internet Explorer（Webブラウザ），Windows Media Player，Windows Messenger，ファイル・ビューア
- IPv6，UPnP，RTC（VoIP），RDP（Thin Client）
- Bluetooth，SDIO，USB，PCMCIA/CardBus
- 日本語IME，手書き入力機能

また，後の節で詳しく解説しますが，本特集のT-Engine，T-Kernelに対応するためのインターフェースも準備しています．

2）プラットホームに応じたOSのカスタマイズ性

Windows CEでは，コンポーネントを組み合わせてOS自体をカスタマイズすることが可能です．また，ビジュアルな開発

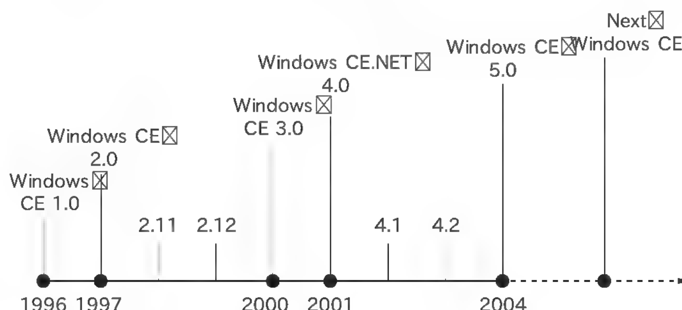


図1　Windows CE製品のロードマップ

環境である Platform Builder も用意されています(図2).

3) アプリケーション開発の柔軟性

デスクトップの開発環境である Visual C++と同等の機能をもった，Embedded Visual C++(eVC)に加え，Visual Studio .NET による，C#，VB.NET を用いた CPU に依存しないアプリケーション，XML Web サービス・クライアントなど多彩なアプリケーション・ソフトウェアの開発が可能になっています.

4) プラットホーム独立性の高いシステム構成

Windows CE はプラットホーム間での移植性を高めるために，ハードウェア依存のカーネル・コードを OEM アダプテーション層(OAL)として独立させています.

プラットホーム開発者は，個々のプラットホームに合わせて初期化時，起動時，実行時にそれぞれ以下の処理を行うフック関数の中身を作成します．カーネルは，ハードウェアに依存した処理が必要なときにこのフック関数を呼び出します.

▶ 初期化時
- プラットホーム・ハードウェア，割り込みなどの初期化

▶ 起動時
- システム・スタートアップ・ルーチン

▶ 実行時
- スケジュール割り込みをカーネルに渡す
- 周辺機器の割り込みをカーネルに渡す
- リアルタイム・クロックへのアクセス
- 電源管理機能
- デバッグ・シリアル，パラレル，および Ethernet 周辺機器へのアクセス
- カーネル I/O 制御処理

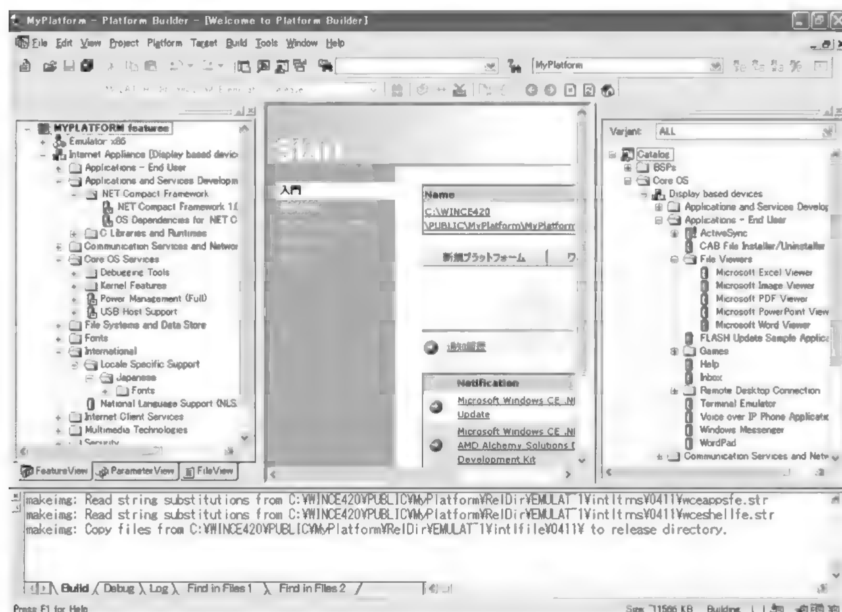


図2 Platform Builder によるビジュアルな開発

本章のメインである Windows CE と T-Kernel を協調動作させる枠組みも，この OAL の考え方を用いており，これら二つの OS 間の実行制御のために，新たに OAL 関数を準備しています．次の節ではこの Windows CE と T-Engine を協調動作させる枠組みについて解説します.

● Windows CE についての誤解と真実

Windows CE は，PDA などの特定用途で用いられる OS と誤解している開発者が見受けられます．そのほかにも，以下のような誤解の例があげられます.

▶誤解1：Windows CE はデスクトップ Windows と同じシステム構成で，組み込み向け OS ではない

—— Windows CE はデスクトップ Windows とは別に設計された OS です．メモリ管理においても組み込み機器向けに CPU に負荷をかけないように設計されています．これはデスクトップ Windows や Linux のように 2～3G バイトのユーザ・プロセス空間を切り替えるのではなく，1G バイトのアドレス空間にプロセスごとに 32M バイトのメモリ・スロットを割り当てています．これにより，コンテキスト切り替えのステップが少なくなっています．Win32 API も組み込みデバイスに必要な機能をサブセットで提供し，OS のサイズを削減しています.

▶誤解2：Windows CE はフット・プリントが大きく，カスタマイズもできないので，組み込み機器に向いていない

—— Windows CE はモジュールごとに機能を選択でき，メモリ使用量を抑えたカスタマイズが可能です．最小構成(カーネルとファイル・システム)で 400K バイト程度で実装が可能です．さらに，TCP/IP，LAN ドライバを含めて 900K バイト程度，デスクトップ・シェル，GUI，日本語フォントを含めても最小 6M バイト程度で構成が可能です.

▶誤解3：オープン・ソースと違って，ソース・コードを参照することができない

——製品版のほか，Web からダウンロードできる評価版でもソース・コードを参照できます．また，ソース・コードはデバッグのために改変し，ビルドを行うことが可能です.

▶誤解4：Windows CE はリアルタイム性がない OS であり，組み込み機器に向いていない

—— Windows CE はリアルタイム OS としての次の機能を備えています.

256 レベルのスレッド優先順位/入れ子(ネスティング)割り込み/スレッド単位のクォンタム設定/プライオリティ・インバージョンの回避/リアルタイム・スレッド優先順位/リアルタイム・スケジューリングのサポート

以上のように Windows CE は組み込み機器に適した OS であり，POS，PDA，PLC(工業用コントローラ)，キオスク端末など，すでに多くの組み込みシステムに用いられています.

COLUMN 1

Windows CE を使ってみよう

Platform Builder の評価版を用いて，実際に Windows CE OS イメージの作成，操作を行ってみましょう．

● 評価版のダウンロードおよびインストール

現行バージョンの Windows CE 4.2（英語版）の評価版は，マイクロソフトの Windows Embedded の Web サイト，

http://www.microsoft.com/japan/windows/embedded/ce.net/

からダウンロードし，Microsoft Windows XP Professional または，Windows 2000 Professional Service Pack 2 のホスト・マシンにインストールできます．

● プロダクト・キーの取得

上記のページの中でダウンロード・サイトへのリンク（図 A）の「Windows CE.NET の最新バージョンを試用」をクリックし表示されるページで，「Windows CE.NET 4.2 評価版をダウンロード」（英語）をクリックすると，図 B のダウンロードのページが表示されます．このページ下部にある，登録サイトへのリンク（図 C），

http://www.microsoft.com/windows/embedded/evalreg

をクリックし，説明に従ってプロダクト・キーを取得してください．

次に，このページの右上にある，「Download」ボタン（図 D）をクリックし，setup.exe の「ファイルのダウンロード」で，「開く」を選択しインストールを開始します．

それでは，Platform Builder に含まれるエミュレータ上で実際に動作する Windows CE を作成，実行してみましょう．

● Platform Builder の操作

1) Platform Builder の起動

［スタート］メニューから［Microsoft Windows CE .NET 4.2］，［Platform Builder 4.2］を選択し，Platform Builder を起動します．

2) プラットホームの構成を設定する

Platform Builder で，［File］メニューの［New Platform］をクリックします．以降，プラットホーム作成ウィザードに従って各種設定を行います（ファイル名，オプションを適当に設定）．

ステップ 1：プラットホーム ワークスペースの作成
ステップ 2：Board Support Packages（BSPs）の選択
ステップ 3：Platform Configuration の選択
ステップ 4：Application & Media 機能の選択
ステップ 5：Networking & Communications 機能の選択
ステップ 6：Completing the New Platform Wizard

3) OS イメージをビルドする

メニュー・バー［Build］で［Build Platform］を選択すると，上で設定したプラットホーム向けの OS イメージのビルドが開始されます（ビルド完了まで，最大，数 10 分程度かかる）．

4) OS イメージをエミュレータ環境にダウンロードする

- ［Target］メニュー，［Configure Remote Connection］を選択し，設定ダイアログ中の，［Download］および［Kernel Transport］の設定をともに「Emulator - 4.20」とし，［OK］をクリックします．
- 次に［Target］，［Download / Initialize］をクリックします．
- Windows CE OS イメージがダウンロードされ，エミュレータ画面上で Windows CE が起動します（図 E）．

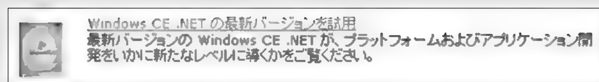


図 A　ダウンロード・サイトへのリンク

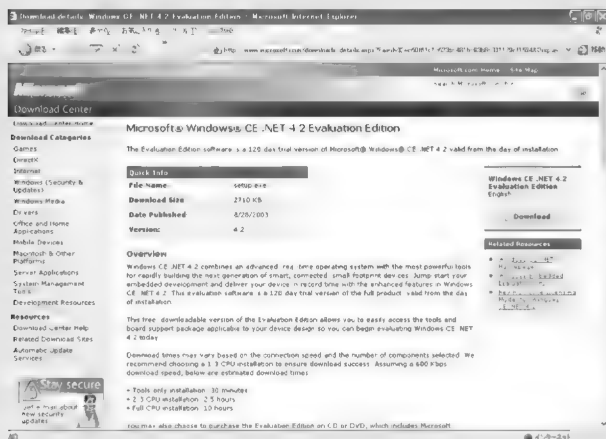


図 B　ダウンロードのページ

IMPORTANT NOTICES:

To register for your product key, please visit:
http://www.microsoft.com/windows/embedded/evalreg.

図 C　登録サイトへのリンク

Windows CE .NET 4.2
Evaluation Edition
English

Download

図 D　Download ボタン



図 E　Windows CE 実行画面

Windows CE/T-Engine ブリッジ・フレームワーク

ブリッジ・フレームワークは，T-Engineハードウェア上にWindows CEとT-Kernelの二つのOSを協調動作させるために，ブリッジ・モジュール，OS間同期通信API，そしてWindows CEのカーネルの新しいフィーチャとして提供される，割り込み・スケジューリング・フック機能から構成されます．ブリッジ・フレームワークの中で，ブリッジ・モジュールのデザインおよび同期通信APIの規定はあくまでマイクロソフトが提案するガイドラインです．ブリッジ・モジュールのT-Kernelへの実装，OS間同期通信APIの実装は，システム全体の開発時に行う必要があります．このブリッジ・フレームワークの最大の目的は，お互いのOSが持っている機能を損なわずに，お互いの資産をそのまま利用可能にすることです．

つまり，このブリッジ・フレームワークを利用することにより両方のプラットホームの利点を生かした，次のようなシステムの構築が可能です．

1) 従来のソフトウェア資産がそのまま流用できる
2) アプリケーションがOS間で協調動作できる

という二つの機能をそれぞれのシステムのパフォーマンスを劣化させず実現することができるようになります．

デスクトップOSと同じ操作環境を実現するのに向いているWindows CEを利用することにより，リッチなUI，ファイル・システム，ネットワークなどのミドルウェア，メディア再生やWebブラウザなどの多岐にわたるアプリケーションが利用可能になります．また，ブリッジ・フレームワークを利用することにより，組み込み業界で幅広いシェアを持つITRONの資産を有効利用し，短期間でさまざまな機能をもつ機器が開発できるようになります．たとえばオート・フォーカスやCCDからの画像転送の処理を過去のITRONの資産を活用し，UIやほかのデバイス(PCを含む)との通信をWindows CEで処理するというような機器に応用できます(そのほかの応用例に関しては表1を参照)．

● 実現方法

今回の目的1)，2)を達成するために，著者らは次のような実装方法を検討しました(この実装方法は，いくつかある方法のうちの一案である)．

実際の実装案の詳細に関しては後述しますが，今回の枠組みではT-KernelとWindows CEは独立したメモリ空間に配置されています．また，デバイスは二つのOSでは共有されません．つまり，各デバイスはどちらか一方のOSでのみ管理されています(タイマ割り込みだけは例外で，双方のOSのスケジューラで使用するため共有している)．このようにデバイスを共有せず，どちらかの一方のOSのみで管理することによりデバイス・ドライバをそのまま利用することが可能です(図3)．

つまり，お互いの資産を活用することが可能になり，ブリッジ・フレームワークを利用するために新たにデバイス・ドライバを開発する必要がありません．一方のOSの管理下であるデバイスをほかのOSから利用する場合には，スレッド/タスク間の同期通信機能を利用することにより利用できるようになります．つまり，それぞれは独立管理なので，一方のOSが管理するデバイスをほかのOSから利用できないということではありません．

最適化された割り込み制御，スケジューラ制御を実現するために次の二つのOAL関数を提供予定です．

1) `pfnOEMIntrOccurs`
2) `pfnOEMReschedule`

詳細は後述しますが，大まかな流れは図4のようになっています．

上記の二つのOAL関数を利用することによりWindows CE，T-Kernel両方のリアルタイム性を同時に確保することが可能になっています．基本的には，それぞれのOSはお互いに独立して動き，お互いの存在を意識することはありません．現在実行すべきOSは外部割り込み，スレッド(タスク)の切り替えを契機に決定します．必要であればOS自体を切り替えることによりお互いのOS・アプリケーションを干渉せずに動作させることが可能です．

また，図5のように割り込み，タスク/スレッドをT-Kernel，Windows CEで同じ優先度で管理することにより，それぞれのOSの利点を生かした柔軟で拡張性の高いシステムの構築を

表1　応用例

機　器	T-Kernel	Windows CE
ディジタル・カメラ	オート・フォーカス，CCDからの画像転送処理	UI，ほかのデバイスとの通信処理
ディジタル・ビデオ・カメラ	光学系，手振れ補正処理	タッチ・パネル，UI，Webアクセス，メールの処理
地上ディジタル・レシーバ，モバイル受信機	チューナ，画像処理	Webアクセス，番組情報取得，ハードディスク・レコーダ処理
多機能プリンタ・システム	PCからの印刷コマンド・データ取得要求，ヘッド制御処理	UI，ネットワーク対応，画像処理
PVR(HDD，DVDレコーダ)	チューナ，画像処理	ストリーミング，ファイル・システム，UPnPによるメディア・コントロール
IP電話	エコー・キャンセラ，CODEC	RTCサーバとの連携，外部デバイス(SD，USB)の利用
ホーム・ゲートウェイ	機器制御，信号処理	TCP/IP，UPnP，IEEE1394
ディジタル・テレビ	TVチューナ，画像処理	リモート・デスクトップ機能，Webアクセス，サーバ機能

可能としています．

つまり，T-Kernelのタスクより高い優先度でWindows CEのスレッドが実行可能です．たとえばWindows CEのネットワーク関係のミドルウェアを利用するためにネットワーク関係のデバイスをWindows CE側で管理した場合でも，Windows CE側のネットワーク・デバイスのパフォーマンスを損なわないシステムを構築することが可能になります．

次の節ではブリッジ・フレームワークの実装を解説します．

カーネル・レベルの動作

ここでは，ブリッジ・フレームワークがどのようにしてWindows CEとT-Kernelを同時に協調動作させているのかを解説します．

● 設計方針

ブリッジ・フレームワークは以下のような方針のもと，設計されました．

1) 従来のソフトウェア資産がそのまま流用できること

それぞれのOSのアプリケーションやデバイス・ドライバなどのソフトウェア資産をそのまま使うことができること．アプリケーションやOSはそれぞれ相手側のOSを意識する必要はない．

2) アプリケーションがOS間で協調動作できること

Windows CEとT-Kernelのアプリケーションが通信や同期を行うためのインターフェースを提供する．

3) システムのパフォーマンスが低下しないこと

リアルタイム性能など，システム全体のパフォーマンスはOS単体で実行した場合と比べても遜色なく動かすことができること．

● 実装概要

ブリッジ・フレームワークは以下のような実装により，上記設計目標を実現しています．

▶メモリ空間を別々に割り当てる

Windows CEとT-Kernelには，それぞれ別の物理メモリ空間を割り当てます．Windows CEの初期化の延長で，T-KernelはWindows CEカーネルの一部として論理空間にマップします(図6)．

アドレス空間は完全に独立しているので，OSはそれぞれの存在を意識する必要はありません．

▶CPU資源の割り当て

それぞれのOSはほかのOSの存在を意識しないので，そのままではほかのOSやそのタスクへCPUを割り当てることができません．そこで本システムでは，ブリッジ・モジュールと呼ぶどちらのOSにも属さないモジュールをカーネル空間に配置して，CPU資源をそれぞれのOSに割りふることで，複数OSの同時実行を可能にしています(図7)．

CPU資源を割りふるタイミングは，割り込み発生時とタスク(Windows CEの場合はスレッド)切り替え発生時です．

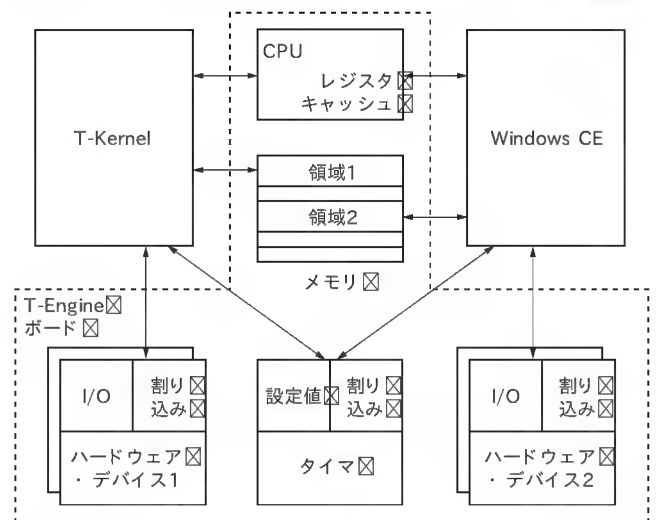


図3 デバイス構成

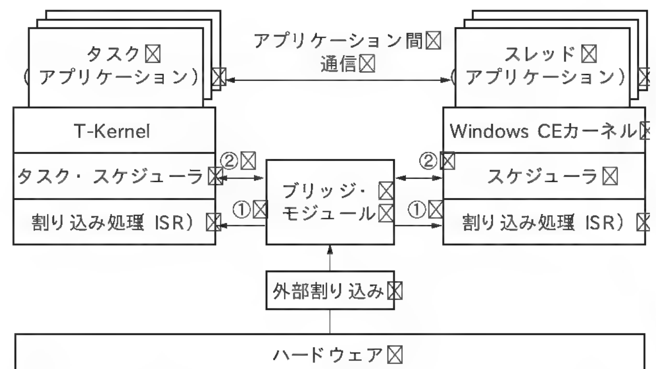


図4 システム構成

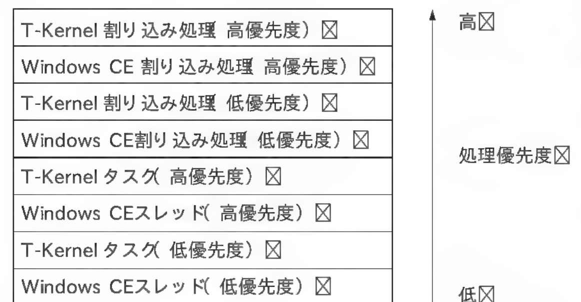


図5 処理優先度

▶割り込み処理のスケジューリング

ブリッジ・モジュールはOSの初期化処理において最初に割り込みを受けられるよう，割り込みベクタ・テーブルを書き換えます(このとき，双方のOSの割り込みベクタ・アドレスを保存しておく)．

割り込みが発生すると，まずブリッジ・モジュールが割り込みを受け取ります．ブリッジ・モジュールは初期化時に保存しておいた割り込みベクタ・アドレスをもとに，それぞれのOS

の割り込み処理を実行して，割り込み元へ復帰します（図8）．

割り込み処理の結果，新たなタスクあるいはスレッドの実行が実行待ちとなる場合があります．そのときの処理は次節で解説します．

▶OSのスケジューリング

割り込みの延長やアプリケーションの同期待ちなどにより新たなタスクやスレッドが実行可能になると，通常はOSのスケジューラが呼ばれ，より優先度の高いタスクが実行されます．

ブリッジ・モジュールは，OSのスケジューラによって新しいタスクが実行されるとき，双方のOSで最高優先度を持つタスク（スレッド）の優先度を比較します．現在実行中のOSより，他OSのタスク（スレッド）の優先度が高い場合，ブリッジ・モジュールは他OSへ制御を移します．これにより，つねにより優先度が高いタスク（スレッド）がCPU資源を使うことができるようにしています（図9）．

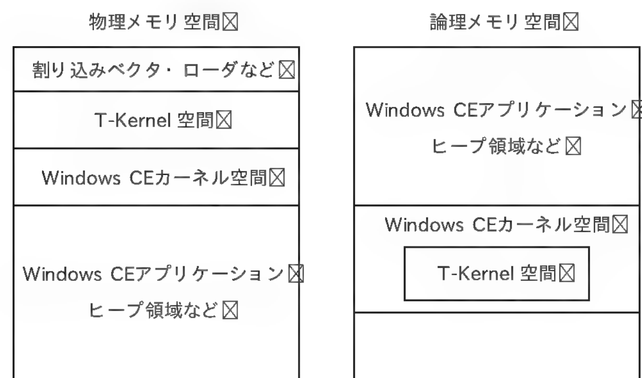


図6　メモリ空間割り当て

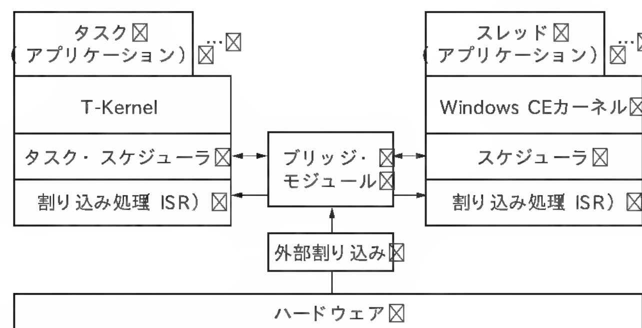


図7　ブリッジ・モジュールとシステム構成

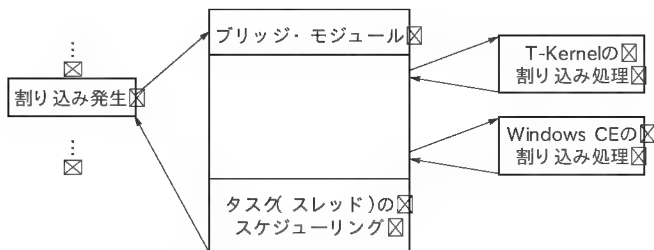


図8　割り込み処理のスケジューリング

なお，ブリッジ・モジュールはOSの切り替えのみを行います．実際のタスクやスレッドのスケジューリングはそれぞれのOSに備わっているスケジューラに任されています．

▶ブリッジ・フレームワークの初期化

OSの初期化の延長で，ブリッジ・フレームワークの実行環境が構築されます．以下のようなシーケンスで初期化が実行されます（図10）．

1) 最初にそれぞれのOSの初期化処理を行います（Windows CE/T-Kernelどちらからスタートしてもよいが，ここではT-Kernelから初期化が開始されるものと仮定する）．
2) OSの初期化が終了したら，ブリッジ・モジュールに制御を移します．ブリッジ・モジュールは割り込みベクタの設定，アドレス・テーブルの作成など，OSのスケジューリングを行うための前準備を行います．
3) 初期設定が終了すると，ブリッジ・モジュールはそれぞれのOSのタスク（スレッド）のスケジューリングを始めます．OSはそれぞれの初期タスクからシステム全体の起動を行っていきます．

▶アドレス・テーブル

アドレス・テーブルは，ブリッジ・モジュールとOS間で互いに呼び出しを行うための処理ルーチンのアドレス・リストです（図11）．割り込みやタスク・スケジューリングなどはこのテーブルに書かれたアドレスをコールすることで制御を移すことができるようになります．

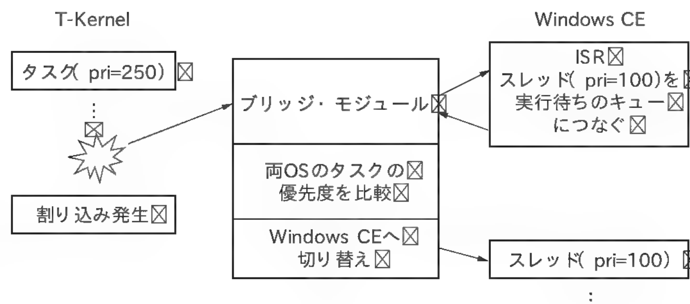


図9　OSのスケジューリング

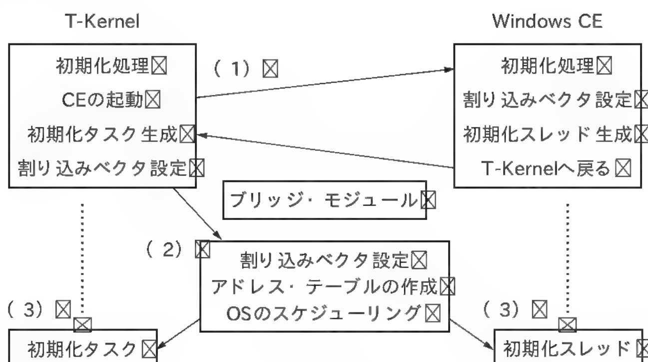


図10　ブリッジ・フレームワークの初期化

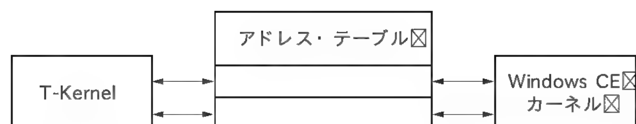


図11　アドレス・テーブル

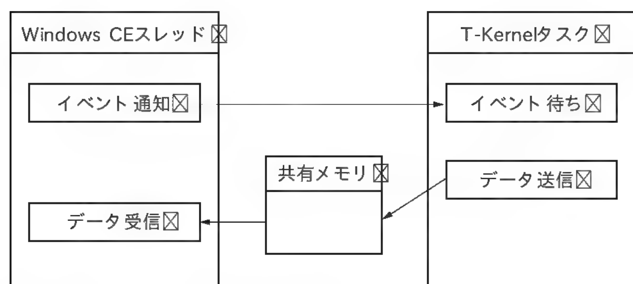


図12　OS間同期通信APIの概念図

アプリケーション・レベルの動作

ブリッジ・フレームワークはWindows CEとT-Kernelのアプリケーションどうしで同期やデータのやりとりなどを行うインタフェース群である「OS間同期通信API」を規定して提案します．

● OS間同期通信API

OS間同期通信APIは，別々のOS上で動作するアプリケーションどうしが協調動作を行うためのAPI群です．

前節で述べたとおり，それぞれのOS上のアプリケーションは，ほかのOSを意識せず従来どおり動きます．Windows CEとT-Kernelのアプリケーションがそれぞれ役割を分担し，お互いに同期を取ったり，データを交換したりしながら協調して動作することができれば，お互いのOSの利点を生かしたシステム設計が可能になります(図12)．

● APIの種類

OS間同期通信APIとして大きく分けて以下の5種類を提案します．

1) 同期機能API

Windows CEとT-Kernelのアプリケーション間で同期を取るためのAPI群です．イベントを待ち合わせたり，通知したりすることができます(図13)．

使い方の例としては，T-KernelからWindows CEに対してイベントを通知する(逆も可)，あるいはWindows CEからT-Kernelへ処理完了を通知するといった使用方法が考えられます．

`XOSCreateEvent`：イベント・オブジェクトの生成
`XOSDeleteEvent`：イベント・オブジェクトの削除
`XOSResetEvent`：イベント待ち解除状態の取り消し
`XOSSetEvent`：イベント待ち状態を解除

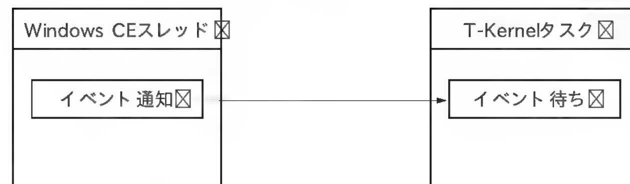


図13　同期機能API

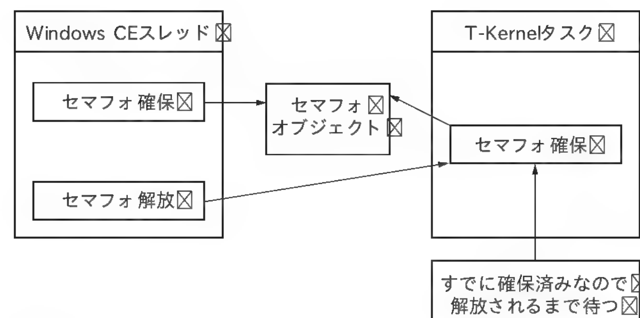


図14　排他機能API

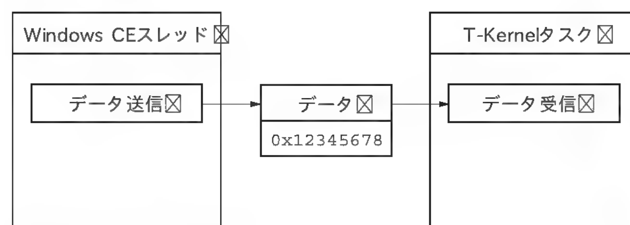


図15　タスク・スレッド間通信API

`XOSWaitForSingleEvent`：イベント待ち状態へ移行
`XOSWaitForMultipleEvents`：複数のイベント待ち

2) 排他機能API

メモリ資源あるいはCPU資源をアプリケーション間で排他的に使うためのAPI群です．一般的にセマフォと呼ばれる機能をOS間で使えるようにしたものです(図14)．

Windows CEとT-Kernel間で同じメモリ空間を使う場合，同時にアクセスされるのを防ぐにはセマフォが有効です．

`XOSCreateSemaphore`：セマフォ・オブジェクトの生成
`XOSDeleteSemaphore`：セマフォ・オブジェクトの削除
`XOSSignalSemaphore`：セマフォへ資源を返却
`XOSWaitSemaphore`：セマフォの資源を確保

3) タスク・スレッド間通信API

アプリケーション間で小さなデータの受け渡しをするためのAPI群です．データの受け渡しとともに，同期を取ることも可能です(図15)．

Windows CEからT-Kernelへイベントを送り，イベント内容を詳細コードで送るといった使い方ができます．

`XOSCreateMailbox`：メールボックス・オブジェクトの生成
`XOSDeleteMailbox`：メールボックス・オブジェクトの削除

COLUMN 2

Windows CE に関する情報源

Windows CE に関する情報源は，以下のものが公開されています．

1) 製品情報

詳細情報については，マイクロソフトの Windows CE .NET の Web サイトを参照してください．

http://www.microsoft.com/japan/windows/embedded/ce.net/

2) 技術情報

Windows CE 向けの技術情報は MSDN ライブラリにも記載されています．

http://www.microsoft.com/japan/msdn/library/（日本語版）

以下の，Embedded 開発/モバイルおよび Embedded 開発/Embedded オペレーティング・システム開発/Windows CE/Windows CE.NET に CE に関する情報があります．

http://msdn.microsoft.com/embedded/（英語版）

3) コミュニティ情報

Windows CE を含めた，Windows Embedded のコミュニティ活動に関する情報は以下のサイトに掲載されています．

http://www.microsoft.com/japan/windows/embedded/community/（日本語版）

http://msdn.microsoft.com/embedded/community/（英語版）

`XOSSendMailbox`：メールボックスへデータを送信

`XOSReceiveMailbox`：メールボックスからデータを受信

4) タスク・スレッド間共有メモリ API

アプリケーション間で通信用に使う共有メモリに関する API 群です．同期機能はありませんが，大きなデータの受け渡しを行う際に適しています（図 16）．

OS 間で画像データをやりとりする場合など，同期機能 API でイベントを送り，データそのものは共有メモリで受け渡しする，といった使い方が考えられます．

`XOSAllocateSharedMemory`：共有メモリの確保

`XOSFreeSharedMemory`：共有メモリの解放

`XOSLockSharedMemory`：共有メモリをページング禁止に

`XOSUnlockSharedMemory`：共有メモリをページング可能に

5) 共通 API

API 共通のエラー・コードを取得する機能です．

`XOSGetLastError`：API の拡張エラー・コードの取得

● API の使用例

ここではイベント・オブジェクトを使って，OS 間の同期を

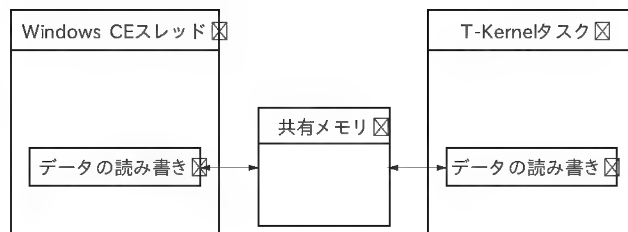


図 16　タスク・スレッド間共有メモリ API

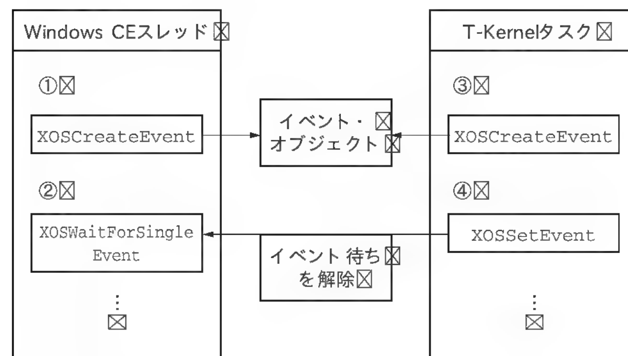


図 17　API 使用例

実現する例を図 17 に示します．

1) Windows CE アプリケーションから `XOSCreateEvent` を発行して，イベント・オブジェクトを作成します．
2) `XOSWaitForSingleEvent` を発行して，T-Kernel アプリケーションからのイベントを待ちます．
3) T-Kernel アプリケーションでは同様に `XOSCreateEvent` を発行します．同一の ID でオブジェクトを作成することで，相手側と同じオブジェクトを使用することができます．
4) `XOSSetEvent` を発行して，Windows CE アプリケーションにイベントを送ります．

おわりに

Windows CE は，最新のマルチメディア，ネットワーキング，多彩なデバイスのサポート，日本語対応など，豊富な機能を同梱し，それらをカスタマイズして製品に利用可能であるという点で，とても柔軟性のある組み込み OS および開発環境です．

さらに，ブリッジ・フレームワークを利用することにより，既存の組み込み OS との共存が可能になります．これにより，それぞれの OS の機能や特徴を損なうことなく，双方の OS の長所を活用する柔軟なシステムの設計が可能になります．また，アプリケーション，デバイス・ドライバなどの既存の資産を有効活用できるため，双方の資産を最大限活用して新しい可能性をもつ機器の開発や設計が容易になります．

しばもと・としひろ/きし・けいいち/おだぎり・やすひろ
マイクロソフト プロダクト ディベロップメント リミテッド
ウィンドウズ開発統括部 クライアント技術開発グループ

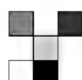
既存の OS を T-Kernel 上でも動作させる

第7章 T-Kernel と Linux のハイブリッド環境による T-Linux の実装

木内 志朗

T-Kernel は ITRON を発展させたものであることからリアルタイム性に優れている．しかし T-Kernel 以外の OS ――たとえば，近年，組み込みで使われるようになってきた Linux は，各種 UNIX の資産がそのまま使用できるという利点がある．そこで T-Kernel と Linux を同時に使い，適材適所でタスクを割りふりたいと考えるのは自然なことだろう．

そこで本章では，割り込みを T-Kernel と Linux とで割りふり，同時に動作させることを可能にしたアーキテクチャ「T-Linux」について解説する．（編集部）



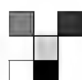
はじめに

現在，組み込みシステム開発において，リアルタイム OS の選択肢として組み込み Linux が採用され，数多くの製品が市場に投入されています．組み込み Linux を採用する理由としては，おもにネットワーク機能や各種アプリケーションが豊富，オープン・ソース，ロイヤリティ・フリーなどが挙げられています．すでに，携帯電話，STB（PVR），ディジタル TV，PDA などをはじめとするディジタル家電製品や，携帯電話の基地局などに代表される通信インフラ分野で Linux が搭載されています．

Linux カーネル 2.6 で追加されたカーネル・プリエンプション機能は，組み込みシステム向けにプロセス応答時間が改善されています．これにより，リアルタイム性は非常に向上していますが，より高いリアルタイム性が必要とされる組み込みシステムも数多く存在します．

たとえば，2004 年末までには，国内でも組み込み Linux を搭載した 3G 携帯電話が市場に投入されます．これらの携帯電話では，GUI，ゲームや PDA 機能といった機能を組み込み Linux を用いた CPU（アプリケーション・プロセッサ）で処理を行い，ベース・バンドという通信機能側には DSP や ITRON などのリアルタイム OS を搭載した別 CPU（ベース・バンド・プロセッサ）が処理を行うという複数のプロセッサを使用した構成となっています．これらを 1 チップの CPU で実現するためにはより高いリアルタイム性が Linux に要求されます．

そこで，T-Linux は，リアルタイム性の高い T-Kernel と Linux とをハイブリッド技術によって融合させることにより，リアルタイム性の高い T-Kernel アプリケーションと Linux のアプリケーションを同時に実装することをめざしています．ここでは，一つの実装例として Linux と T-Kernel のハイブリッド環境について紹介します．



T-Linux の実装方式

T-Linux の実装方式については，いろいろと検討しましたが，Linux 側の修正を最小限にすることを前提にハイブリッド環境を採用しました．Linux カーネルおよび Linux で動作するプロセス全体を T-Kernel の一つのタスク（Linux タスク）としています．Linux タスクの優先順位は任意に設定できますが，T-Kernel 側のタスクのリアルタイム性を確保するために，一番低い優先順位に設定しています．T-Kernel のスケジューラがシステム全体（T-Linux）のスケジューリングを行いますが，Linux 側にもスケジューラがあり，Linux プロセスをスケジューリングしていることを意識する必要があります（図 1）．

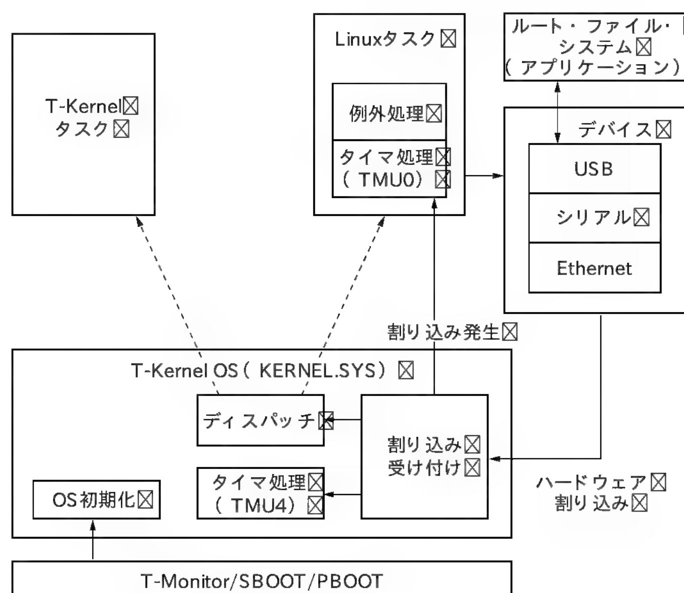


図 1　T-Linux の構成

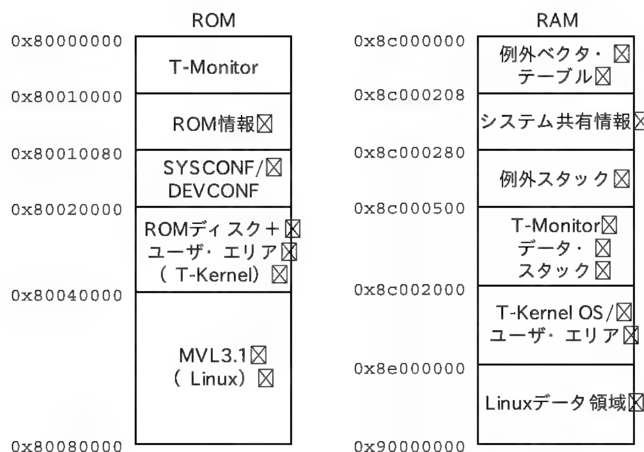


図2　メモリ・マップ

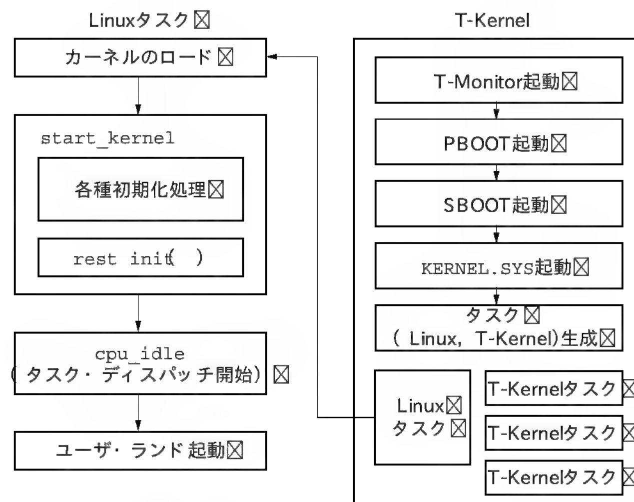


図3　T-Linux起動手順

割り込みについては，T-Kernel側で使用する割り込みとLinux側で使用するデバイスからの割り込みを分離することにより，T-Kernel側のリアルタイム性を損なわないようにしています．T-Kernelは，現在，T-Engineフォーラムより公開されているソース・コードを使用しており，T-Kernel側ではMMU制御を行いません．MMU制御はLinuxカーネル側で行っています．したがって，T-KernelおよびT-Kernelタスクは，Linuxのカーネル空間で動作します．

● 実装環境

モンタビスタ ソフトウェア ジャパン(株)で行ったT-KernelとLinuxのハイブリッド環境によるT-Linuxの実装にあたっては次のような環境を使用しました．

- ターゲット：T-Engine/SH7751R
- T-Kernel：T-Engineフォーラムより公開されているソース・コードを使用
- Linux：MontaVista Linux Pro 3.1/SH-4(リトルエンディアン)
- Linuxデバイス・ドライバ：LAN，タイマ，USBストレージ(ルート・ファイル・システム)

Linuxのルート・ファイル・システムは，T-Engineオンボードのフラッシュ・メモリ，PCMCIAメモリ・カード，USBストレージなどが想定されますが，今回の実装ではUSBストレージをルート・ファイル・システムとして使用しています．深い理由はないのですが，オンボードのフラッシュ・メモリではサイズが十分でなく，またPCMCIAカード・スロットは，無線LANやモデム・カードなどでの使用が考えられるため，USBストレージをルート・ファイル・システムとして利用しました．

今回の実装では，T-Kernel側とLinux側でイメージが完全に独立しているので，開発ツールはそれぞれの開発ツールを使用しています．T-Kernel，LinuxともにGNUツール・チェーンが標準的なツールなので，コンパイラの違いによる問題はありません．

● メモリ構成

T-Kernel側で使用するメモリ空間とLinux側で使用するメモリ空間を完全に分離しています．T-Kernelで使用しない領域をLinux側のメモリ領域にしています．Linux側のデータ領域は，LinuxのMMU機能によりページ単位で管理が行われ，Linuxプロセスなどは，この領域にロードされ，実行されます．T-Engine/SH7751Rは8Mバイトのフラッシュ・メモリ，64MバイトのSDRAMが実装されています．それぞれのメモリ・マップを図2に示します．

● ブート処理

T-KernelとT-Kernelから起動されるLinuxのコードは完全に独立しており，それぞれT-Engineのフラッシュ・メモリに独立したイメージで書き込まれています．ブート処理は次のとおりです．

電源投入後，T-Engineに搭載されているT-Monitorからブートされます．T-MonitorによりT-Kernelが起動されます．T-Kernelの初期化終了後，イニシャル・タスクとしてLinuxタスクを生成します．Linuxタスクは，LinuxカーネルをRAM上に展開し，Linuxのイニシャル処理である`start_kernel`にジャンプします．これにより，Linuxが起動します．

Linuxの初期化処理は，通常のLinuxの処理と同じであり，デバイス・ドライバの初期化を含めた各種初期設定を行い，ルート・ファイル・システムをマウントします(図3)．

● SH固有のバンク・レジスタ

SH-4におけるT-KernelやLinuxは，SH-4バンク1レジスタをそれぞれ表1と表2のように実装しています．ハイブリッド環境では，これらのバンク1レジスタを二つのOSで競合しな

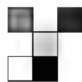
いように保持する必要があります．そこで，T-Linuxでは次のようにバンク1レジスタの割り当てを行っています．特にLinuxでの割り込み処理はおもにアセンブラで実装されており，レジスタ不足などから作業用としてバンク1を使用しているため，T-Linuxでは一時スタックに退避させ，作業用で使用した後に復帰させなければなりません．表3にT-Linux実装前後での割り当てを示します．

● 割り込み処理

割り込み処理については，T-Kernel側で処理する割り込みとLinux側で処理する割り込みを内容により振り分けています．割り込みが発生するとT-Kernelの割り込みベクタに設定されているそれぞれの処理に分岐します．T-Kernel側で使用するデバイスからのハードウェア割り込みについては，そのままT-Kernelの割り込み処理を実行します．それ以外の割り込みについては，Linux側の割り込み処理の中で再度分岐しています(図4)．

T-LinuxではT-Kernelの割り込みベクタ・テーブル(0x8c000000)をそのまま使用しています．Linux初期化時に，Linuxで使用する割り込みハンドラをT-Kernelの割り込みベクタ・テーブルに再設定しています．割り込みベクタ登録は，Linux初期化処理で，T-Kernelシステム・コール`tk_def_int()`を使用することにより行っています．Linux側で使用する割り込みベクタを表4に示します．

SH-4のT-KernelのシステムコールとLinuxのシステム・コールは，無条件トラップ(`TRAPA`)によって発生しますが，`TRAPA`レジスタの条件によりT-Kernel，Linuxそれぞれのシステム・コールを識別することが可能です．`TRAPA`例外レジスタ値の条件でT-Kernel，Linuxシステム・コールを識別し，それぞれの処理を行っています(表5)．



今後の課題

今回のT-Linuxの実装では，T-KernelとLinuxの二つのOSをハイブリッド環境により実現しています．基本的には，割り込みの振り分け処理を変更していますが，それぞれのOS自体は変更していないので，T-Kernel，Linuxアプリケーションは，変更する必要なく動作します．割り込み振り分け処理は，完全にCPUアーキテクチャに依存するため，ほかのアーキテクチャに実装する場合は，二つのOSでのレジスタ使用方法も含めて検討する必要があります．

Linux側でもともと持っている割り込みマスク処理により，システム全体の割り込みがマスクされ，結果としてT-Kernel側のリアルタイム性に影響が出る場合も考えられます．今後の課題としてT-Kernel側のリアルタイム性を確保するために，Linux側のデバイス・ドライバやカーネルの割り込みマスク処理を修正する必要があるかもしれません．この点については今後の課題として検討しています．

表1　T-Kernelバンク1レジスタ割り当て

レジスタ	用　途	レジスタ	用　途
R0	作業用	R4	作業用
R1	予約	R5	予約
R2	動作モード・レジスタ	R6	例外スタック・ポインタ
R3	システム・スタック・トップ	R7	例外作業用

表2　Linuxバンク1レジスタ割り当て

レジスタ	用　途	レジスタ	用　途
R0	作業用(割り込み)	R4	作業用(割り込み)
R1	作業用(割り込み)	R5	作業用(割り込み)
R2	割り込み要因	R6	割り込みマスク
R3	作業用(割り込み)	R7	CURRENTポインタ

表3　T-Linuxバンク1レジスタ/拡張変数割り当て

レジスタ	用　途	レジスタ	用　途
R0(共通)	作業用	R4(共通)	作業用
R1(Linux)	CURRENTポインタ	R5(Linux)	割り込みマスク
R2(T-Kernel)	動作モード・レジスタ	R6(T-Kernel)	例外スタック・ポインタ
R3(T-Kernel)	システム・スタック・トップ	R7(共通)	作業用
変　数	用　途		
int_reg2(Linux)	割り込み要因		

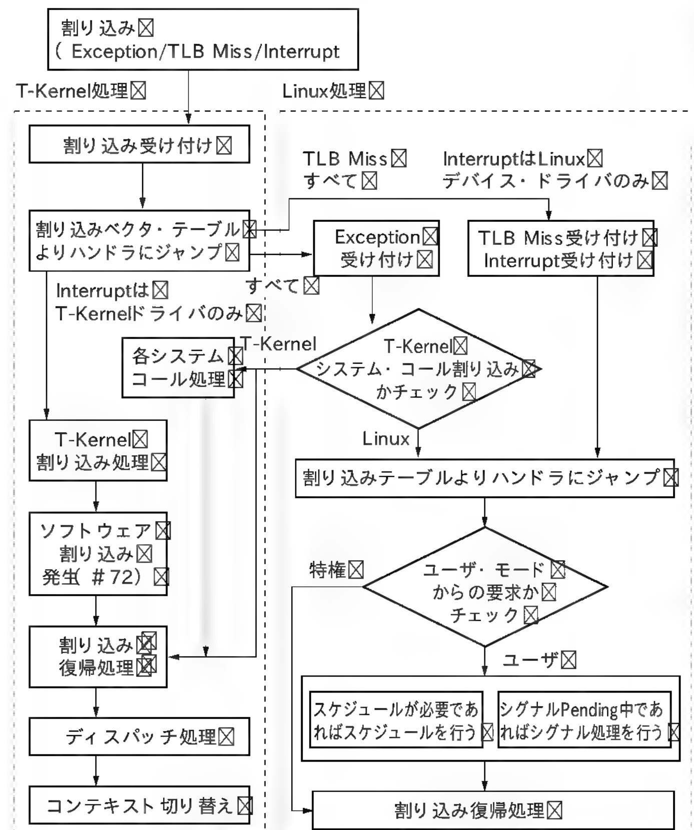


図4　割り込み処理

表4　Linux IRQ番号

割り込み要因	ベクタ・アドレス	例外コード	詳細内容
例外割り込み(vbr+0x100) general_exception	0x8c000010	0x80	初期ページ書き込み例外
	0x8c000014	0xa0	TLB保護例外(読み出し)
	0x8c000018	0xc0	TLB保護例外(書き込み)
	0x8c00001c	0xe0	CPUアドレス・エラー(読み出し)
	0x8c000020	0x100	CPUアドレス・エラー(書き込み)
	0x8c000024	0x120	FPU例外
	0x8c00002c	0x160	無条件トラップ(TRAPA命令)
	0x8c000030	0x180	一般不当命令例外
	0x8c000034	0x1a0	スロット不当命令例外
	0x8c000100	0x800	FPU割り込み
	0x8c000104	0x820	FPU割り込み
TLBミス割り込み(vbr+0x400) tlb_miss	0x8c000008	0x40	TLBミス例外(読み出し)
	0x8c00000c	0x60	TLBミス例外(書き込み)
ハードウェア割り込み(vbr+0x600) interrupt	0x8c000040	0x200	拡張ボードEthernet割り込み
	0x8c000044	0x204	拡張ボードUART割り込み
	0x8c000064	0x320	USB割り込み
	0x8c000080	0x400	タイマ(TMU0)割り込み

表5　TRAPAのレジスタ値
これ以外のTRAPA例外レジスタ値の場合，Linuxシステム・コール処理を行う

レジスタ値	内　容
0x70	T-Monitorサービス・コール
0x71	T-Kernelシステム・コール・拡張SVC call_entry()
0x72	ハードウェア割り込み復帰システム・コール_tk_ret_int()
0x73	タスク・ディスパッチャ呼び出し dispatch_entry()
0x74	予備
0x75	SRレジスタ・ロード機能 load_SR()
0x76	デバッガ・サポート機能 call_dbgspt()
0x77	プロセス強制終了要求(非サポート)

T-KernelおよびT-Kernelタスクは，Linuxのカーネル空間で動作しています．そのため，T-Kernelタスクに対しては，MMUによるメモリ保護が行われています．T-Kernelタスクの誤った処理により，Linux側のアドレス空間にアクセスしないように注意する必要があります．

現在の実装では，Linuxカーネル側からT-Kernelのシステム・コールを発行することが可能なため，二つのOS間のデータの受け渡しや同期については，Linux内にてT-Kernelのシステム・コールを使用することにより実現可能です．しかし，T-KernelタスクとLinuxプロセス間での情報を通知するAPIをそれぞれに作る必要があります．現在は独自のAPIを作っていますが，T-Kernel，Linuxそれぞれに標準的なAPIを規定する必要があります．

二つのスケジューラが動作しているため，システム全体としては，T-Kernel側で行う処理とLinux側で行う処理を注意深く考える必要があります．T-Kernel側では本当にリアルタイム性が要求される一部のタスクのみを実装し，それ以外の処理についてはLinux側で行うようにしたほうが，問題が発生しにくいと思います．

おわりに

本記事では，T-Linuxの実装例としてT-KernelとLinuxのハイブリッド環境について紹介しました．T-Linuxの実装は，まだ開発途上のものであり実用化に向け今後も開発を続けています．2004年7月7日～9日に行われる「第7回 組み込みシステム開発技術展」にて参考出品する予定です．

きうち・しろう　モンタビスタ ソフトウェア ジャパン(株)

各社のT-Engineボードの実際(1)

APPENDIX 1

ルネサスマイコン搭載 T-Engineのラインナップ

山田 浩之

ルネサス製マイコン搭載T-Engineのラインナップ

ルネサステクノロジは，ユビキタス・ネットワーク環境を実現するプラットホームとして**図1**に示すようなコントローラ系やプロセッサ系の32ビットRISCマイコンを中心にして，T-Engineの開発を進めています．

表1にすでに開発キットとしてリリース(または近日リリース)のT-Engine/μT-Engineの仕様を示します．2002年8月に第一弾としてSH7727版T-Engineをリリース．すぐにM32104版μT-Engineをリリースし，その後にSH7751R版T-Engineと，SH7760版T-Engineのリリースを行い，近々にSH7145版μT-Engineをリリース予定です．

また，これらルネサスマイコンを搭載したT-Engine/μT-Engineは累計で1500セット以上を出荷し，1000件近いプロジェクトで検討しています．

今回は，最近リリースを行ったSH7760 T-Engineのハードウェアを中心に解説します．

SH7760 T-Engineを使うための基礎知識

● SH7760の概要

図2にSH7760のブロック図を示します．SH7760は，テレマティクス機器などの車載情報機器やPOS端末の産業機器向けにSHマイコンのCPUコアSH-4とLCDコントローラ，USBホスト，音声やサウンド，車載LAN(CAN)，メモリ・カードなどの多様なインターフェースを1チップに集積した高性能プロセッサです．このた

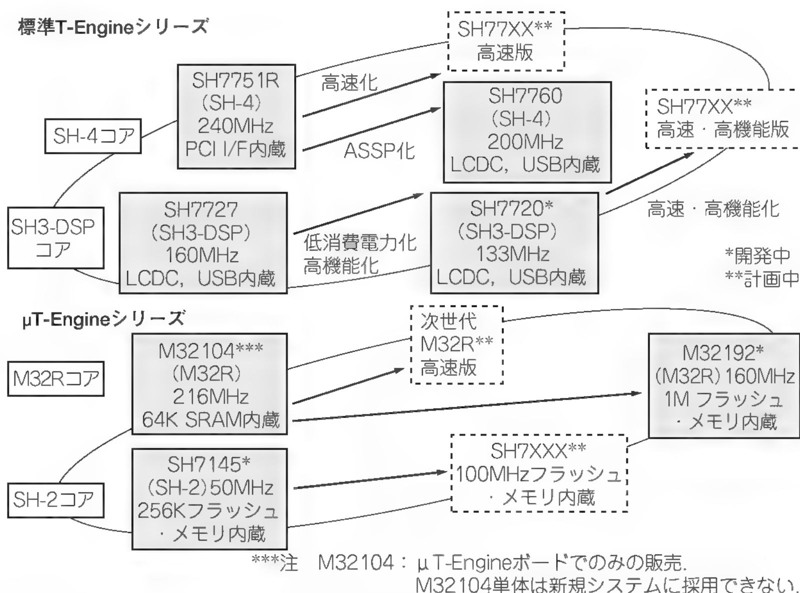


図1　T-Engine対応マイコンのロードマップ

表1　ルネサス製T-Engine概略仕様

(価格は消費税込み)

	T-Engine			μT-Engine	
	SH7727 T-Engine	SH7751R T-Engine	SH7760 T-Engine	SH7145 μT-Engine	M32104 μT-Engine
用途	PDAなどの携帯端末	ホーム・ゲートウェイ	CISやPOS端末	プリンタ，FA	SoCプラットホーム
CPU	SH7727 (SH3-DSP) 96MHz (130MIPS) LCDC，USB，etc	SH7751R (SH-4) 240MHz (430MIPS) PCI I/F，etc	SH7760 (SH-4) 200MHz (360MIPS) CAN，LCDC，etc	SH7145 (SH-2) 50MHz (65MIPS) ROM 256Kバイト， RAM 8Kバイト	M32104 (M32R) 216MHz (243MIPS) キャッシュ8Kバイト， SRAM64Kバイト
メモリ	SDRAM 32Mバイト， フラッシュ・メモリ 8Mバイト	SDRAM 64Mバイト， フラッシュ・メモリ 8Mバイト	SDRAM 64Mバイト， フラッシュ・メモリ 8Mバイト	SRAM 1Mバイト， フラッシュ・メモリ 1Mバイト	SDRAM 16Mバイト， フラッシュ・メモリ 4Mバイト
I/F	PCMCIA I/F LCD (320 × 240) I/F USBホスト 拡張バスI/F シリアルI/F	PCMCIA I/F LCD (320 × 240) I/F USBホスト 拡張バスI/F シリアルI/F	PCMCIA I/F LCD (320 × 240) I/F USBホスト 拡張バスI/F シリアルI/F	CF Card I/F SD Card I/F eTRON I/F シリアルI/F	CF Card I/F MMC I/F eTRON I/F シリアルI/F
拡張バス	SHバス (01-01)	PCIバス (04-01)	SHバス (04-01)	SHコントローラ・バス (01-02)	M32Rバス (02-02)
標準価格	￥145,000	￥204,750	￥204,750	未定	￥157,500
拡張ボード	LANボード ユニバーサル・ボード マルチメディア・ボード ARカメラ・ボード	LANボード ユニバーサル・ボード マルチメディア・ボード	LANボード ユニバーサル・ボード マルチメディア・ボード ARカメラ・ボード	LANボード ユニバーサル・ボード ARカメラ・ボード	LANボード Bluetoothボード ARカメラ・ボード ユニバーサル・ボード

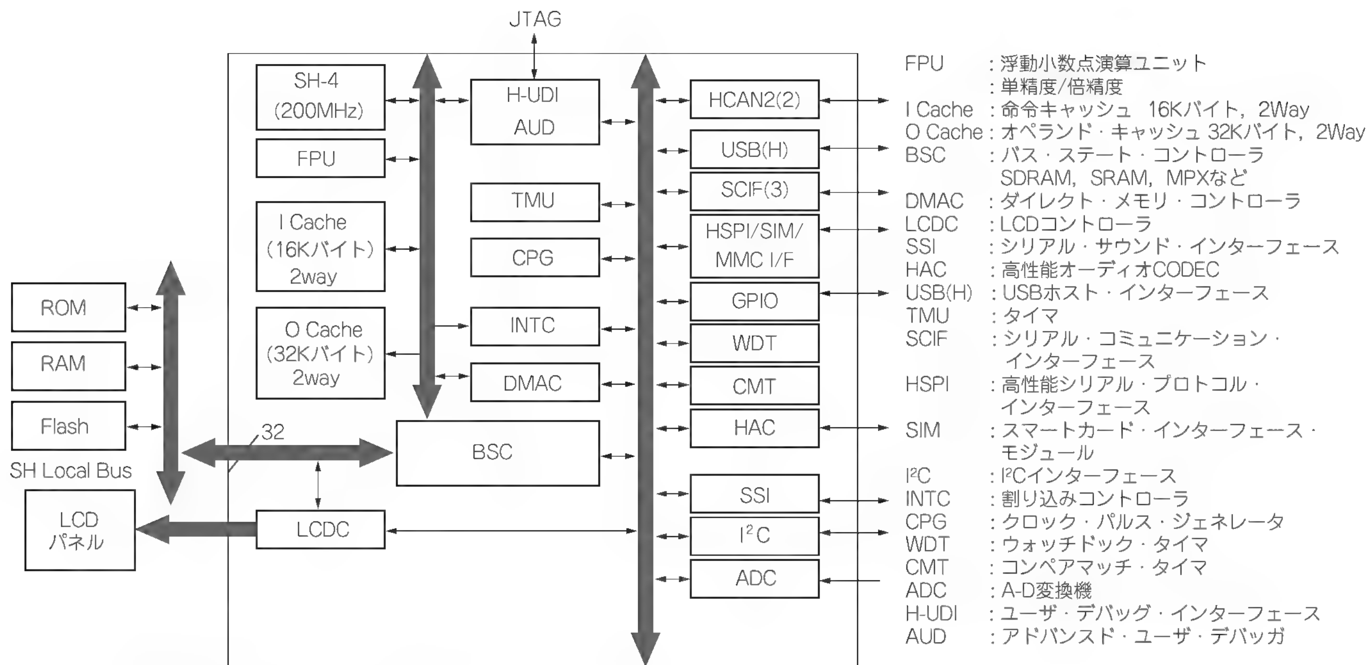


図2　SH7760のブロック図



写真1　SH7760 T-Engineの外観

め，1チップで高性能なテレマティクス端末やPOS端末などの産業機器や民生機器を実現することが可能です．また，SH-4はMMUを搭載しているため，T-Kernel Extensionを利用した仮想記憶などのより高度なプログラミングが可能です．

● SH7760 T-Engineの概要

写真1にSH7760 T-Engineの外観を示します．SH7760 T-EngineはCPUボード，QVGAのタッチパネル付きLCDボード，デバック・ボード，I/Oボードで構成されています．このデバック・ボードには，E10AなどのJTAG ICEを接続するためのコネクタやT-Engineに搭載されているフラッシュ・メモリにプログラムを書き込むための簡易モニタが書かれたEPROMが搭載されています．またI/OボードはCANなどのT-Engine規格以外のSH7760内蔵機能がスルー・ホールとして引き出されているため，これらの機能を評価することが可能です．

● SH7760 T-Engineの各部詳細

図3にSH7760 T-Engine CPUボード(以下，本T-Engine)のブロック図，**写真2**にその各部を示します．本T-EngineはSH7760のローカル・バスに32ビット・バス幅で32MバイトSDRAMを2個(64Mバイト)が，16ビット・バス幅で8Mバイトのフラッシュ・メモリ，PCMCIAコントローラなどが接続されています．標準T-Engineハードウェアに必要なインターフェース機能のほとんどはSH7760に内蔵されている機能で実現しています．以下に標準T-Engineに必要なインターフェース機能を中心に各部の機能について解説します．

▶スイッチ：④〜⑧

標準T-Engineに必要な電源スイッチ，リセット・スイッチ，NMIスイッチと，本T-Engine独自の8ビット・ディップ・スイッチ，システム・リセット・スイッチが搭載されています．電源スイッチは0.5s以上押すとON，電源ON状態で2s以上押すとOFFになり，8ビット・ディップ・スイッチはSH7760のポート端子，動作モード端子に接続されています．また電源ON条件も設定することができます．システム・リセット・スイッチは，本ボードに搭載されているすべてのデバイスをリセットすることが可能です．

▶PCMCIA：⑨⑩

SHマイコンのバスに直結可能な丸文(株)製PCMCIAコントローラ(MR-SHPC-01 V2)を搭載しています．本コントローラはSH7760に16ビット・バス幅で接続され，PC Card Standard97標準規格に準拠したカードとインターフェースするコントローラで5.0V/3.3Vに対応しています．コントローラの割り込みはSIQ3〜SIQ0の4本あり，SH7760への入力はIRLコードで入力されます．

詳細はMR-SHPC-01 V2のマニュアルを参照してください．

`http://www2.marubun.co.jp/`

▶USBホスト・インターフェース：⑪

SH7760内蔵USBコントローラを使用しています．内蔵コントローラはOpenHCI(Open Host Controller Interface)1.0レジスタ・セッ

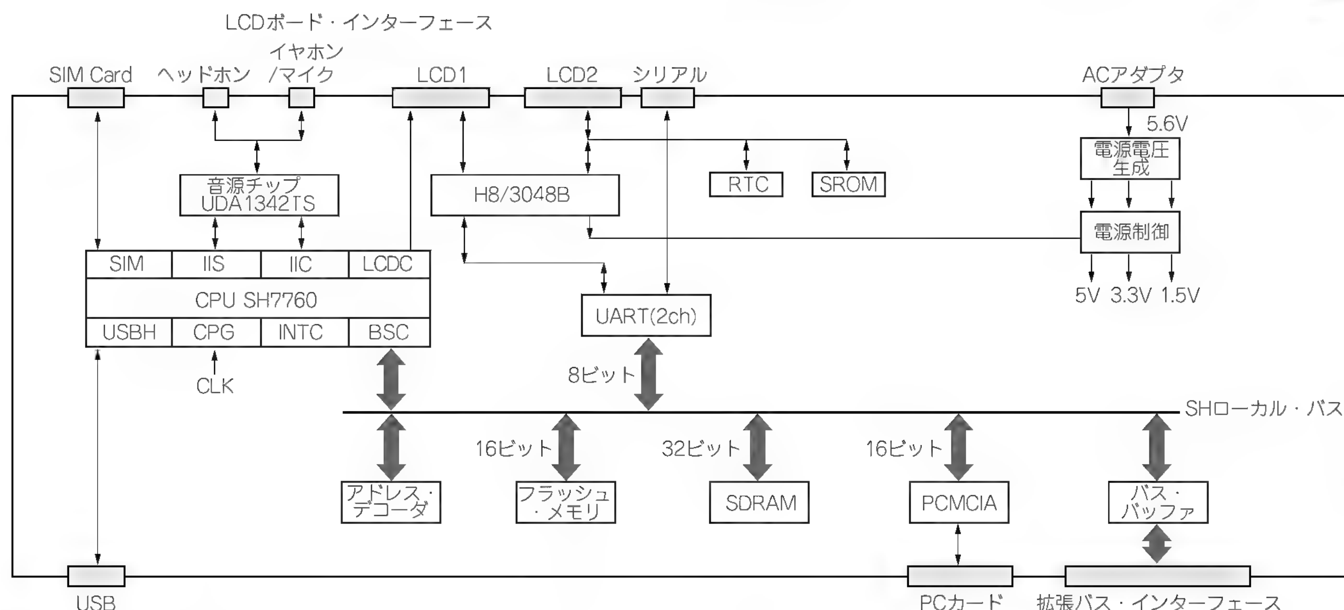
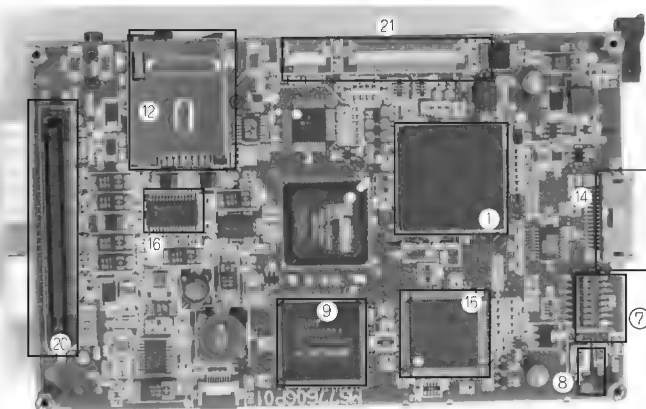
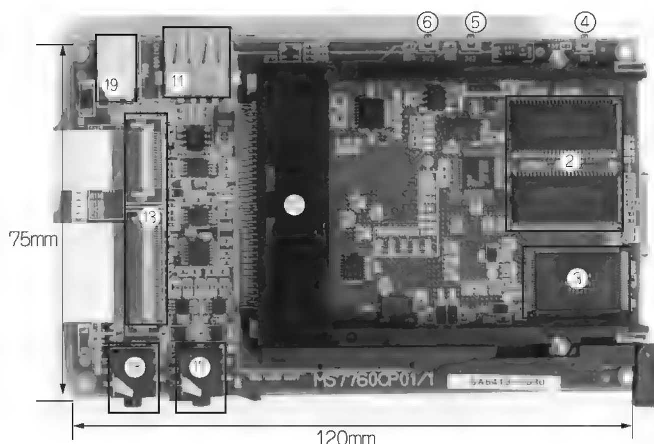


図3　SH7760 T-Engineのブロック図





①CPU:SH7760　②SDRAM(32Mバイト×2)　③フラッシュ・メモリ(8Mバイト)　④パワー・スイッチ　⑤リセット・スイッチ　⑥NMIスイッチ　⑦8ビット・ディップ・スイッチ　⑧システム・リセット・スイッチ　⑨PCMCIAコントローラ　⑩PCMCIAコネクタ　⑪USBホスト・インターフェース　⑫eTRONカード・インターフェース　⑬LCDインターフェース　⑭シリアル・ポート・インターフェース　⑮電源コントローラ　⑯音声CODEC　⑰ヘッドホン・コネクタ　⑱イヤホン・マイク・コネクタ　⑲ACアダプタ　⑳拡張コネクタ　㉑SH7760 I/Oコネクタ

写真2　SH7760 T-Engineの各部

ト，USB Ver1.1をサポートしてます．転送モードはフル・スピードとロー・スピードをサポートし，4種類の転送モードをサポートしてます(コントロール転送，バルク転送，インタラプト転送，アシンクロナス転送)．

▶eTRONカード・インターフェース(SIMカード・インターフェース)：⑫

SH7760内蔵のSIMインターフェース・モジュールを使用しています．

▶LCDインターフェース：⑬

SH7760内蔵のLCDコントローラを使用しています．本ボードに同梱されているLCDボード(QVGA：240×320，16ビットRGB，TFT)に接続され，SH7760のエリア3に接続されたSDRAMの一部を表示用メモリとして使用しています．また，タッチ・パネルは電源コントローラ(H8/3048)に接続されており，この電源コントローラの割り込みとレジスタで読み取ることができます．

▶シリアル・ポート・インターフェース：⑭

図4にシリアル・インターフェース，および電源コントローラ・ブロックを示します．SH7760のバスに16550互換のコントローラが接続されており，このChAが電源コントローラにChBがシリアル・ポート・インターフェースに接続されています．コントローラからの割り込みは，ChAはSH7760のIRL9に，ChBはIRL11に接続されています．

▶リアルタイム・クロック

電源コントローラに接続されており，電気二重層コンデンサによ

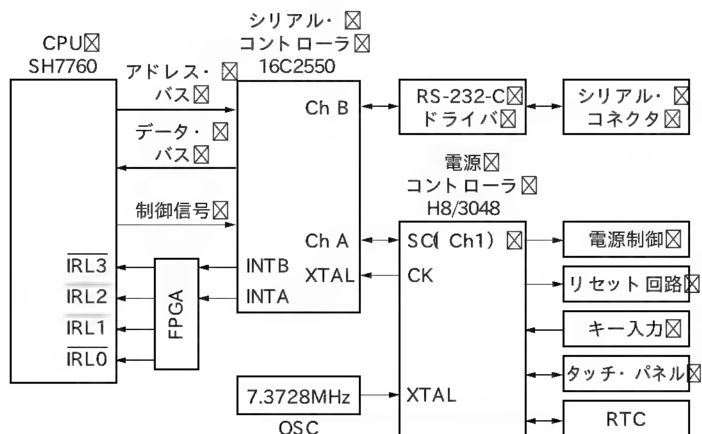


図4　シリアル・インターフェース，および電源コントローラのブロック図

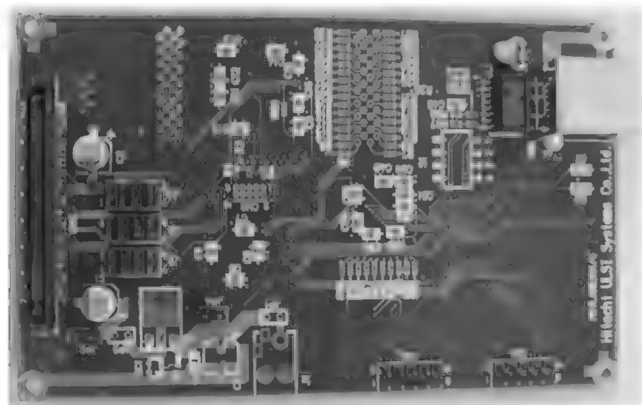


写真3　SH7760 LAN拡張ボード

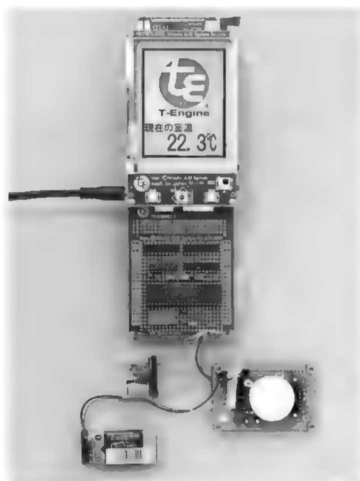


写真4
ユニバーサル・ボードの使用例

る電源バックアップ機能を搭載しています．

▶ 音声CODEC：⑯⑰⑱

SH7760内蔵のシリアル・サウンド・インターフェース（SSI）を使用しています．SSIはクロック同期のシリアル・インターフェースで，設定によりクロックとデータの極性/位相を変更できます．また，チャネルに分割されるさまざまなシリアル・オーディオ・ストリームの送受信が可能です．オーディオCODECはPhilips社製（UDA1342TS）を搭載し，ヘッドホン・コネクタ（ステレオ出力），イヤホン・コネクタ（マイク入力，イヤホン出力）に接続されてます．オーディオCODECの制御にはSH7760内蔵のI^{2}Cインターフェースを使用しています．電子ボリュームも搭載されており，電源コントローラのレジスタを制御することにより音声出力時にボリューム調整が可能です．

▶ 拡張コネクタ・インターフェース：⑳

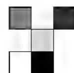
T-Engine規格の拡張バス専用コネクタを搭載しています（京セラエルコ社製 140ピン・コネクタ）．

拡張バス・インターフェースはSH7760のデータ・バス，アドレス・バス，制御信号などが出力されてます．

（T-Engine規格の拡張バスの種類に関しては，コラムを参照）

▶ SH7760 I/Oコネクタ：㉑

本コネクタは，T-Engine規格には必要ないのですが，SH7760の内蔵モジュール機能を評価するために搭載してます．具体的には，SH7760の内蔵モジュールの端子をフレキシブル・コネクタ経由で引き出し，I/Oボード（本ボードに同梱）にスルー・ホールで出力しています．信号端子は，HCAN2×2チャネル，SCIF×2チャネル，I^{2}C×1チャネル，A-Dコンバータ×4チャネル，CMTです．ただし，コネクタは未実装なので，外部端子に接続する場合は，スルー・ホールに直接接続するか，コネクタを実装してください．



T-Engine拡張ボード

SH7760T-Engineの拡張バスにはSHのバスが出力されており（キーイング01-01），すでにリリースされているSH7727の拡張ボードを共通に使用することができます．現在SHバスの拡張ボードは数種類が提供され，ユーザはこれらの拡張ボードを使用することで，さまざまなシステムの評価が可能です．

● SH7760 LAN拡張ボード

写真3にLAN拡張ボードを示します．LANコントローラ（1チャネル）とシリアル・コントローラ（2チャネル）を搭載した拡張ボードです．LANコントローラにはSMSC社製LAN91C111-EXを搭載し，100BASE-TX/10BASE-Tに対応しています．

● SH7760 FPGA拡張ボード

搭載されている拡張コネクタはすべてのT-Engineに接続可能な00-00のキーイングを搭載しています．FPGAはAltera社のCyclone（EP1C20F400）が搭載されています．FPGAのほかには，LEDやSDRAMや各種I/Oを接続するための汎用I/Oピンが搭載されています．これにより，ユーザは独自の回路やI/Oを接続することが可能です．

〔問い合わせ先：（株）アルティマ マーケティング統括部マーケティング2部 TEL：045-476-2155　http://altimanet.com/〕

● ユニバーサル・ボード

拡張ボードを自作する場合に便利な4種類のユニバーサル・ボードがセットになってます．SHバスのキーイング（01-01）のほかにPCIバス（04-00）やSHコントローラ系バス（02-01），M32バス（02-02）などの拡張コネクタが搭載されているため，SH7751R, SH7727, M32R104, SH7145, V_R5500, V_R4131, TX4956のT-Engine/μT-Engineに接続可能です．ユニバーサル・ボード上に周辺LSIを接続し，それらを応用したシステムのソフトウェアやハードウェアを効率よく開発できます．写真4は秋月電子通商で購入した温度センサ・

COLUMN

T-Engine拡張コネクタの種類

組み込みシステムの場合，標準化されたバスを使うよりもCPU独自のバスにそのまま周辺LSIを接続したほうがパフォーマンスが向上する場合があります．このため，T-Engienフォーラムで専用コネクタを規格化しています．この拡張コネクタの特徴はまちがってほかのCPUのバスに拡張ボードを接続してしまって，ボードを壊してしまわないように，コネクタの両端に誤挿入キーイングがついています．**図A**に現在規定されている各社のキーイングを示します．これらのキーイングは，拡張コネクタを提供している京セラエルコ社のコネクタ型名「10 5603 14 XX-XX 861」の「XX-XX」の箇所をそれぞれのバスに割り当てています．

考え方としては，

XX-XXの最初のXXがバス・タイプを定義しています．具体的には，SH：01，M32R：02，ARM：03，PCI：04，FR：05です．

最後のXXはメーカのコードを定義しており，CPUタイプを定義しています．SH：01，SHコントローラ系：02，ARM：03，V_R：04，FR：05です．また，ORを取れるようなキーイングもあり，00となってます．このため，00-00がすべてのT-Engineに接続可能で電源ボードなどの用途に，04-00がPCIタイプのすべてのT-Engineに接続する用途に使用できます．

コネクタ寸法などの詳細は京セラエルコ社のWebページ，http://www.kyocera-elco.com/を参照してください．

プラグ	name	レセプタクル
10 5603 14 XXXX 861	Parts No.	20 5603 14 XXXX 861
Top VIEW LARGE SMALL	side view	Top VIEW SMALL LARGE
KEY−A KEY−B	type	KEY−B KEY−A
	01-01 RENESAS SH	
2	02-02 RENESAS M32R	
	03-03 YOKOGAWA ARM	
	04-04 NEC V_R(PCI)	
	05-05 FUJITSU	

プラグ	name	レセプタクル
10 5603 14 XXXX 861	Parts No.	20 5603 14 XXXX 861
Top VIEW LARGE SMALL	side view	Top VIEW SMALL LARGE
KEY−A KEY−B	type	KEY−B KEY−A
	05-01 FUJITSU FR	
	04-01 RENESAS SH(PCI)	
	04-00 PCI-Bus	
	00-00 MASTER	
	01-02 RENESAS SH (シングル)	

図A　T-Engine拡張コネクタ・キーイング

キットと人感センサ・キットをユニバーサル基板上に搭載してSH7760 T-Engineで周辺温度を測定している例です（秋月電子通商：http://akizukidenshi.com/）．

おわりに

今回，紹介したように，SH7760 T-Engineと各種拡張ボードを使用することで，すぐに，ある程度のハードウェアを構築することが可能です．また，T-Engine上のデバイス・ドライバやミドルウェアを使用することで，短期間でユーザのシステムの評価やソフトウェア開発が可能です．また，これらハードウェアの回路図や部品リストは本ボードに同梱されており，これらを参考にユーザ独自のハードウェアを構築することができるので，ぜひ，開発にT-Engineを活用してみてはいかがでしょうか．

参考文献

(1)（株）ルネサス テクノロジ ホームページ，http://www.renesas.com/jpn/
(2)（株）ルネサス テクノロジ T-Engineホームページ，http://www.renesas.com/jpn/products/mpumcu/tools/tengine/index.html
(3) 大和矢 千恵，SH7760 T-Engine，TRONWARE Vol84.，パーソナルメディア
(4) 越戸 孝司，SH7760概要とマイクロアーキテクチャ，TRONWARE Vol.84，パーソナルメディア

やまだ・ひろゆき　（株）ルネサスソリューションズ

各社のT-Engineボードの実際(2)

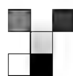
APPENDIX 2

ARM9を2個搭載したマルチCPU T-Engine

桜井 厚

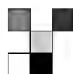
はじめに

富士通(株)が開発したMB87Q1100は，ARM9を2個搭載したマルチCPUチップです．MB87Q1100の評価ボードであるT-Engineボードを横河ディジタルコンピュータ(株)と共同開発したので，紹介します．



ARM9プロセッサとは

1998年にARM社が開発したARM7の上位に位置付けられるプロセッサがARM9ファミリです．富士通がライセンスを受けているARM9はARM926EJ-SおよびARM946E-Sです．ARM926EJ-SはJavaコードの高速処理を可能にするJazelle技術を搭載しており，MMUを搭載しているため，携帯電話やPDAに向いたアプリケーション・プロセッサです．また，MMUのないARM946E-Sはハードディスクや DVDレコーダなどの組み込み制御機器に向いたコントローラです．



非対称マルチCPUシステム

ASICを使っている顧客の要求として，最近はARMのマルチCPUが増えています．たとえばARM7TDMIをシステム・コントローラに，ARM946E-Sをサーボ制御に使うASICがあります．このような非対称のマルチCPUシステムは，じつは組み込みでは特殊なものではなく，携帯電話などにも一般的に見られるものです．これは組み込み機器が，人やコンピュータのようなインテリジェントなものとのインターフェース部分，デバイスとのインターフェース部分という二面性をもつ性格からであると考えられます．

このような処理は単一のCPUを用いてリアルタイム・マルチタスクで処理することができます．しかし，携帯電話，ハードディスクに見られるように，性能や電力への要求が厳しく，タスクがある程度固定されている組み込み用途では非対称マルチCPUによるタスク分割が多く用いられています．

このような理由から，ARM926EJ-SとARM946E-Sという性格の異なるCPUコアを搭載した，非対称のマルチCPUシステムのSoCプラットホームと，それを搭載したLSIであるMB87Q1100を開発しました(写真1)．

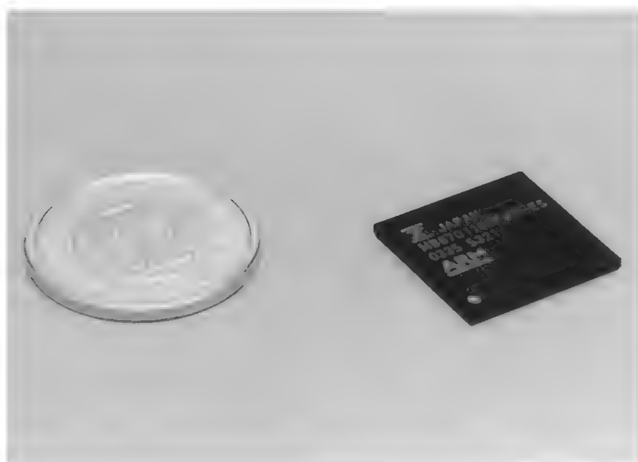
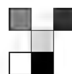


写真1　MB87Q1100

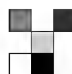


MB87Q1100の特徴

本LSIは，前項で紹介した，異なる性格のARM926EJ-SとARM946E-Sの二つのCPUを搭載しています．本LSIはマルチCPUシステムのバス性能の向上のために，マルチレイアAHBを採用しています．また，本LSIは両CPUが同時に動作可能なデュアルCPUモードと，片方のみのCPUが動作するシングルCPUモードをサポートします．これにより，非対称マルチCPUシステムの評価だけでなく，シングルCPUシステムの評価も可能になっています．本LSIの仕様を表1に示します．

図1にMB87Q1100のブロック図を示します．本CPUの大きな特徴として，内部バスであるAHBLiteを外部に拡張できる点があります．これは，本CPUがSoCのプラットホームとして使用されることを前提に設計されており，外部にFPGAなどを接続し，そこにAMBAモジュールを搭載して評価することを目的にしているためです．

これらのシステムはSoCとして1チップにインテグレートされることを想定しています．SRAMを接続する非同期バスだけでなく，内部バスであるAHBLiteを外に出しているのは，これを可能にするためです．



T-Engineボード

MB87Q1100を使った標準T-Engineを写真2に，仕様を表2に示します．

パーソナルメディア(株)から発売されている「T-Engine/ARM926-MB8開発キット」には前項で紹介したARM9マルチCPUであるMB87Q1100搭載の標準T-Engineボードのほか，リアルタイムOS「T-Kernel」，開発用基本ミドルウェア，GNU開発環境，仕様書などのドキュメント類が含まれており，本キットと開発用のPCだけで，T-Engine上のミドルウェアやアプリケーションの開発が可能です．

詳しくはT-EngineのWebサイト(http://www.t-

表1　MB87Q1100の仕様

項　目	説　明
プロセス	CMOS 0.11 μm プロセス
パッケージ	FBGA 400(15×15mm)
最大動作周波数	CPU：200MHz　AHB：100MHz　APB：50MHz
CPU	ARM926EJ-S，ARM946E-Sを搭載
キャッシュ	命令：16Kバイト，データ：16Kバイト(両CPUとも)
TCM(Tightly Coupled Memory)	命令：64Kバイト，データ：32Kバイト(両CPUとも)
動作モード	デュアルCPUモード，シングルCPUモード
バス	マルチレイアAHB，APB×2
外部拡張AHB	AHBLite外部拡張機能を内蔵し，外部にAHBのマスタおよびスレーブ・モジュールを接続可
クロック制御機能	●CPU，AHB，APBのおのおのに対して基準クロックに対する周波数比を設定することができるギア機能 ●CPUのみの動作を止めるスタンバイ・モード ●LSI内のすべてのクロックを止めるストップ・モード
メモリ・コントローラ	SRAM，フラッシュ用に8本，SDRAM用に1本のチップ・セレクト
DMA	転送モードはブロック，バースト，デマンド，ビートをサポートしており，8チャネル搭載
その他周辺	●割り込みコントローラ ●UART：2チャネル ●タイマ：2チャネル×2(各APB上にそれぞれ接続) ●GPIO
消費電力	450mW(デュアルCPUモード時，CPU：200MHz動作時，Typical条件)

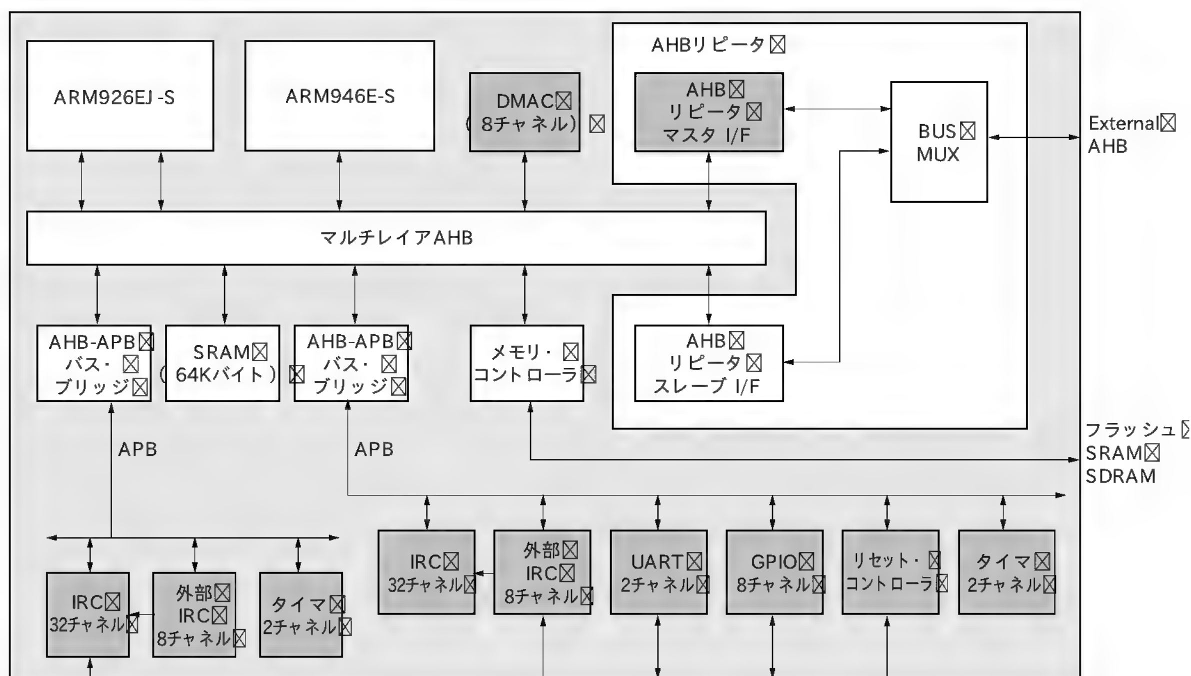
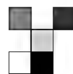


図1　MB87Q1100のブロック図

engine.org/)を参照してください．



T-Engineボードでの実機デバッグ

本ボードの特徴として，MB87Q1100の内部バス(AHBLite)拡張コネクタを搭載していること，ETMコネクタを搭載していることがあげられます．

特にETMコネクタはARMの標準コネクタであり，ARM社をはじめとしたさまざまなベンダから提供されているICE/Traceを接続できます．富士通で動作確認したICEはARM製RealView ICE/Trace，および横河ディジタルコンピュータ製Advice Pocketです．

ARM9プロセッサの実機デバッグ手段としては基本であるJTAG-ICEと，さらにオプションのTraceがあります．JTAG-ICEはJTAGポートからARM9コアを制御し，ステップ実行，レジスタ表示，メモリ表示などを行うことができます．Traceは内部のETM(Embedded Trace Macrocell)で圧縮されたARM9コアのトレース情報(アドレス情報，データ情報の時系列による変化情報)を表示，ブレークすることができます．写真3，図2に実機デバッグのよう

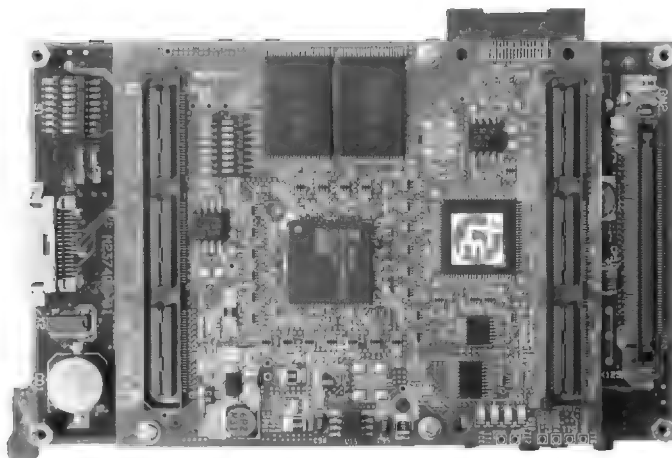


写真2　ARM9 T-Engineボード



写真3　デバッグの実際

表2　T-Engineボードの概略の仕様

項　目	説　明
CPU	富士通製ARM926EJ-S/ARM946E-SデュアルCPU
動作周波数	200MHz
フラッシュ・メモリ	16Mバイト
SDRAM	64Mバイト
入出力I/F	USB(ホスト)，PCMCIAカード，シリアル，eTRONチップ，ヘッドホン出力，マイク入力，拡張バスI/F，AHB-Lite I/F
そのほかの機能	RTC
電源	ACアダプタ
外形寸法	120×75mm(突起物を除く)

すを示します．

組み込み開発環境ではソフトウェアが動作しない場合，実際のシステムに組み込まれた環境でデバッグすることが非常に重要になるため，ICEが必須となります．

さらにチップ内部にETMを搭載すれば，図3，図4のようにリアルタイム・トレース情報が得られるため，デバッグ効率が非常に向上します．

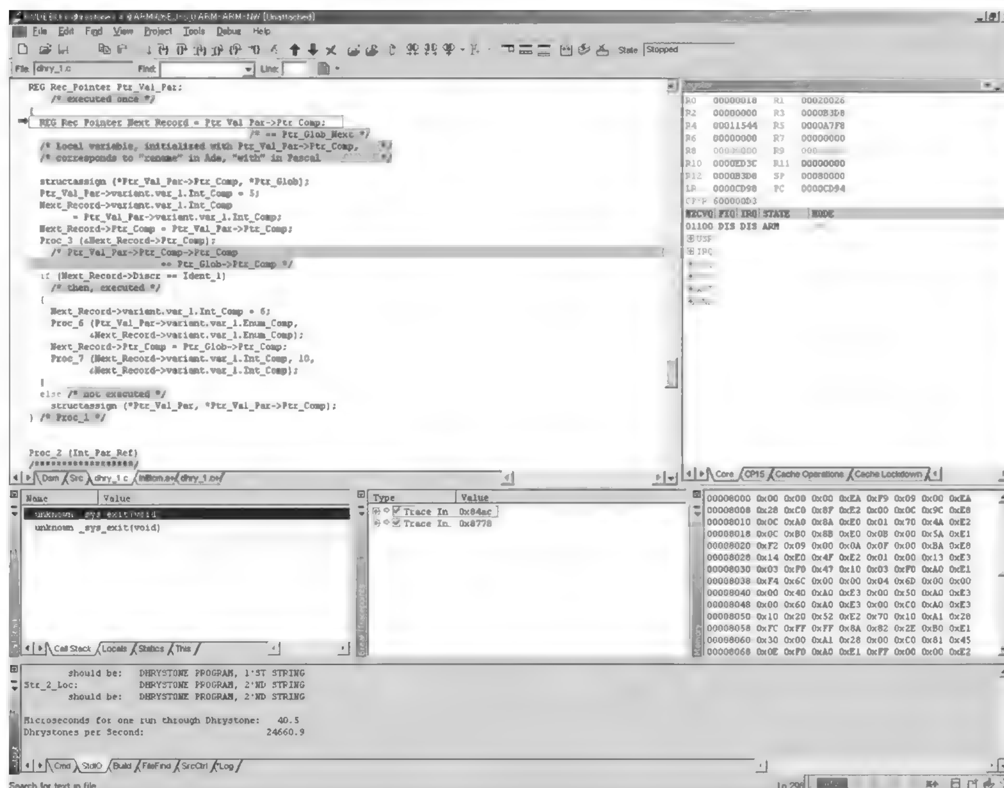


図2　デバッガ画面

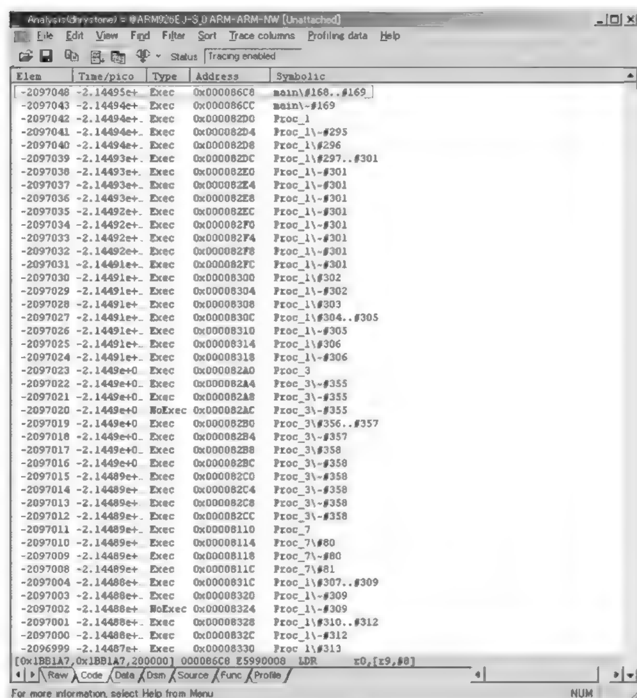


図3　リアルタイム・トレースのようす

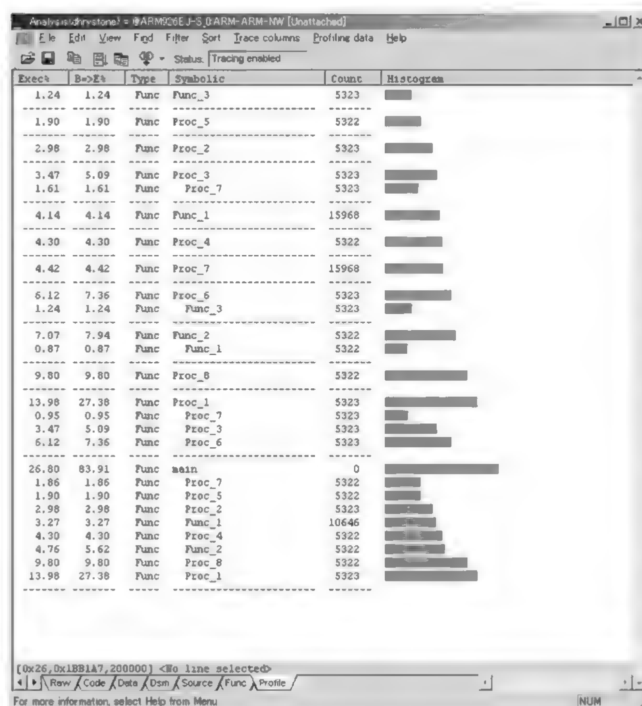


図4　トレース情報解析

将来の方向

T-Engineボードはソフトウェア開発の標準プラットホームとなるだけでなく，ハードウェア開発の標準プラットホームとなる可能性を秘めています．

特に近年のFPGAの大容量化にともない，最近では検証の一部として必須となってきています．富士通はその検討も進めており，別の機会に紹介できればと考えています．

参考文献，URL
(1)「強者の方程式」，日経エレクトロニクス，2002/1/14
(2) T-Engineとは，パーソナルメディア，http://www.personal-media.co.jp/
(3) T-Engineボード，横河ディジタルコンピュータ，http://www.ydc.co.jp/emb/index.html

さくらい・あつし　富士通(株)



第17回

GCC2.95から追加変更のあったオプションの補足と検証(その5)

岸 哲夫

今回も引き続きgcc2.95から追加変更のあったオプションの補足と検証を行います．特に「最適化オプション」について扱います．

（筆者）

● -ffinite-math-only

結果が「非数値」あるいは「無限」ではないと仮定して浮動小数点演算のための最適化を行うオプションです．

この場合の「非数値」はNAN（not a numberの意味）です．0.0/0.0の値をNANと名付けます．

ソースと生成されたコードをリスト1～リスト3に示します．

リスト1 浮動小数点演算のための最適化を行う例（test213.c）

```
//結果が「非数値」あるいは「無限」ではないと仮定して浮動小数点演算のための最適化を
行う例
#define _GNU_SOURCE
#include <stdio.h>
#include <math.h>
const float f1 = 3.1212312312312312f;
const float f2 = 6.5432165432165432f;
float    x;
float    y;

float func(float a)
{
    x    =    10;
    x    =    NAN  *    x;
    y    =    x    * f1;
    return a * f1 / f2;
}
```

最適化の結果，関数内の変数x，yともに直接NAN値がセットされています．むだな計算は排除されています．

ちなみに，NANはC99規格で規定されています．GCC2.95では使用できませんでしたが，現バージョンでは使用できます．ソースを見てわかるように，_GNU_SOURCEを定義することによってGNUライブラリの拡張関数の宣言が有効になります．

● -fforce-addr

このオプションを付加すると，メモリ中のアドレス定数を，算術演算で使用する前にレジスタにコピーします．このオプションにより，処理速度が多少速くなる場合があります．

ソースと生成されたコードをリスト4～リスト6に示します．

オプション付きで生成されたソースを見ると，printfを呼び出す前にレジスタaxに値をセットしてから処理しています．一方，オプションなしで生成されたソースでは，レジスタが指しているアドレスの内容を使用しています．

メモリ中の数値を演算するよりもレジスタに格納された数値を演算するほうが，もちろん速くなります．このオプションは試してみる価値があると思います．

リスト2 結果が「非数値」あるいは「無限」ではないと仮定して浮動小数点演算のための最適化を行うオプションを付けて生成されたアセンブラ・ソース（test213a.s）

```
    .file     "test213.c"
.globl f1
    .section  .rodata
    .align 4
    .type     f1, @object
    .size     f1, 4
f1:
    .long     1078444609
.globl f2
    .align 4
    .type     f2, @object
    .size     f2, 4
f2:
    .long     1087463944
    .align 4
.LC1:
    .long     2143289344
    .text
.globl func
    .type     func, @function
func:
    pushl     %ebp
    movl      %esp, %ebp
    movl      $0x41200000, %eax
    movl      %eax, x
    flds      x
    flds      .LC1
    fmulp     %st, %st(1)
    fstps     x
    flds      x
    fmuls     f1
    fstps     y
    flds      8(%ebp)
    fmuls     f1
    fdivs     f2
    leave
    ret
    .size     func, .-func
    .comm     x,4,4
    .comm     y,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 3　オプションなしで生成されたアセンブラ・ソース(test213.s)

```
    .file      "test213.c"
.globl f1
    .section  .rodata
    .align 4
    .type     f1, @object
    .size     f1, 4
f1:
    .long     1078444609
.globl f2
    .align 4
    .type     f2, @object
    .size     f2, 4
f2:
    .long     1087463944
    .section  .rodata.cst4,"aM",@progbits,4
    .align 4
.LC2:
    .long     1078444609
    .align 4
.LC3:
    .long     1087463944
    .text
    .p2align 2,,3
.globl func
    .type     func, @function
func:
    pushl     %ebp
    movl      %esp, %ebp
    flds      8(%ebp)
    movl      $0x7fc00000, %eax
    fmuls     .LC2
    fdivs     .LC3
    movl      %eax, x
    movl      %eax, y
    leave
    ret
    .size     func, .-func
    .comm     x,4,4
    .comm     y,4,4
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

なお，-fforce-memオプションは，メモリ・オペランドについて算術演算を行う前に，そのメモリ・オペランドをレジスタに強制的にコピーして，すべてのメモリ参照を潜在的な共通部分式とします．しかし，この処理は可搬性の面では有害で，最適化とはいえないように思えます(リスト 7)．

● -fomit-frame-pointer

このオプションを付加すると，フレーム・ポインタを必要としない関数の場合，フレーム・ポインタをレジスタにもちません．これにより，フレーム・ポインタをセーブ，設定，リストアする命令をなくすことができます．結果として多くの関数で利用可能なレジスタが一つ増えます．問題はGDBによるデバッグが不可能になってしまう点です．

リスト 4　メモリ中のアドレス定数を，算術演算で使用する前にレジスタにコピーする例(test214.c)

```
//メモリ中のアドレス定数を、算術演算で使用する前にレジスタにコピーする例
#include <stdio.h>
int main()
{
    int p = 1;
    long   adr = (long)&p;
    adr  = adr+2;
    printf ("%x¥n",adr);
    adr  = adr+2;
    printf ("%x¥n",adr);
    adr  = adr+2;
    printf ("%x¥n",adr);
    adr  = adr+2;
    printf ("%x¥n",adr);
    return 0;
}
```

リスト 5　オプションを付けて生成されたアセンブラ・ソース(test214a.s)

```
    .file      "test214.c"
    .section  .rodata
.LC0:
    .string   "%x¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $1, -4(%ebp)
    leal      -4(%ebp), %eax
    movl      %eax, -8(%ebp)
    movl      -8(%ebp), %eax
    movl      %eax, -8(%ebp)
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    movl      $.LC0, %eax
    pushl     %eax
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    movl      $.LC0, %eax
    pushl         %eax
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    movl      $.LC0, %eax
    pushl     %eax
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    movl      $.LC0, %eax
    pushl     %eax
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 6　オプションなしで生成されたアセンブラ・ソース(test214.s)

```
    .file     "test214.c"
    .section  .rodata
.LC0:
    .string   "%x¥n"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    movl      $1, -4(%ebp)
    leal      -4(%ebp), %eax
    movl      %eax, -8(%ebp)
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    leal      -8(%ebp), %eax
    addl      $2, (%eax)
    subl      $8, %esp
    pushl     -8(%ebp)
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 7　-fforce-memオプションを付けて生成されたアセンブラ・ソース(test215a.s)

```
    .file     "test215.c"
    .section  .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string   "%x¥n"
    .text
    .p2align 2,,3
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    subl      $8, %esp
    leal      -2(%ebp), %eax
    pushl     %eax
    pushl     $.LC0
    movl      $1, -4(%ebp)
    call      printf
    popl      %edx
    popl      %ecx
    pushl     %ebp
    pushl     $.LC0
    call      printf
    popl      %edx
    popl      %ecx
    leal      2(%ebp), %eax
    pushl     %eax
    pushl     $.LC0
    call      printf
    popl      %edx
    popl      %ecx
    leal      4(%ebp), %eax
    pushl     %eax
    pushl     $.LC0
    call      printf
    xorl      %eax, %eax
    leave
    ret
    .size     main, .-main
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 8　フレーム・ポインタを使用しない例(test216.c)

```
//フレームポインタをレジスタに持たない例
#include <stdio.h>
void sub1(void);
void sub2(void);
void sub3(void);
int main()
{
    sub1();
    sub2();
    sub3();
    return 0;
}
void sub1(void)
{
    int  a;
    int  b;
    a    =    10;
    b    =    20;
    return;
}
void sub2(void)
{
    int  a;
    int  b;
    a    =    11;
    b    =    21;
    return;
}
void sub3(void)
{
    int  a;
    int  b;
    a    =    12;
    b    =    22;
    return;
}
```

GDBによるデバッグ作業中は，ユーザ・プログラムが関数呼び出しを行うたびに，その呼び出しに関する情報が生成されます．その情報には，ユーザ・プログラム内においてその呼び出しが発生した場所，関数呼び出しの引き数，呼び出された関数内部のローカル変数などが含まれます．その情報がスタック・フレームと呼ばれるデータ・ブロックに保存されます．そのデータを参照するためにフレーム・ポインタが必要となります．

ソースと生成されたコードをリスト 8～リスト 10に示します．

このように関数の出口と入口で行っている処理が省略されています．もちろん，GDBを使う際には，このオプションを付けないほうがわかりやすいと思います．

● -fschedule-insns

ターゲット・マシン上でこのフラグがサポートされている場合，必要なデータを利用可能になるまで待つことによる実行の

リスト 9　オプションを付けて生成されたアセンブラ・ソース(test216a.s)

```
    .file     "test216.c"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    call      sub1
    call      sub2
    call      sub3
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
.globl sub1
    .type     sub1, @function
sub1:
    subl      $8, %esp
    movl      $10, 4(%esp)
    movl      $20, (%esp)
    addl      $8, %esp
    ret
    .size     sub1, .-sub1
.globl sub2
    .type     sub2, @function
sub2:
    subl      $8, %esp
    movl      $11, 4(%esp)
    movl      $21, (%esp)
    addl      $8, %esp
    ret
    .size     sub2, .-sub2
.globl sub3
    .type     sub3, @function
sub3:
    subl      $8, %esp
    movl      $12, 4(%esp)
    movl      $22, (%esp)
    addl      $8, %esp
    ret
    .size     sub3, .-sub3
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

リスト 10　オプションなしで生成されたアセンブラ・ソース(test216.s)

```
    .file     "test216.c"
    .text
.globl main
    .type     main, @function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    andl      $-16, %esp
    movl      $0, %eax
    subl      %eax, %esp
    call      sub1
    call      sub2
    call      sub3
    movl      $0, %eax
    leave
    ret
    .size     main, .-main
.globl sub1
    .type     sub1, @function
sub1:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    movl      $10, -4(%ebp)
    movl      $20, -8(%ebp)
    leave
    ret
    .size     sub1, .-sub1
.globl sub2
    .type     sub2, @function
sub2:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    movl      $11, -4(%ebp)
    movl      $21, -8(%ebp)
    leave
    ret
    .size     sub2, .-sub2
.globl sub3
    .type     sub3, @function
sub3:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    movl      $12, -4(%ebp)
    movl      $22, -8(%ebp)
    leave
    ret
    .size     sub3, .-sub3
    .section  .note.GNU-stack,"",@progbits
    .ident    "GCC: (GNU) 3.3.3 20040412 (Red Hat Linux 3.3.3-7)"
```

遅延を防ぐために，命令の順番の変更を行います．これは遅い浮動小数点命令やメモリ読み込み命令の実行において，それらの結果を必要とする命令の前にほかの命令を詰め込みます．

ただし，この処理を行うと，デバッグ時に混乱すると思います．しかし，完成版をコンパイルするときに使用すれば，速度の高速化が図れるはずです．

● -fschedule-insns2

前のオプションと似ています．実行の遅延を防ぐためにレジスタに数値を割り当てるといった最適化を行います．やはりデバッグが難しくなるでしょう．

きし・てつお

オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第8回 サービス・コールの概要・その5

岸田 昌巳

シミュレーション環境を使いながら各サービス・コールについて説明してきましたが，今回はシステム状態管理機能の解説を行います．

今回はハードウェアに近いということもあり，サービス・コールの解説にとどめます．今後の連載では，実際のハードウェアを利用した演習へと続くので，その中で各CPUごとのサービス・コール実装の違いなどの解説も予定しています．細かなCPUごとの違いに関しては，今後の解説をご期待ください．

システム状態管理機能

TOPPERS/JSPカーネルでは，システム管理機能として表1に示す10項目と，μITRON4.0仕様にはない項目を一つ提供しています．

ここでいう「システム」とは，カーネルとアプリケーションも含めたものを指し，これらのサービス・コールでアプリケーションからシステム全体を操作することができます．

サービス・コールとして提供している機能は，カーネルの状態を把握したり，ディスパッチの許可禁止，CPUのロックなどです．大まかには「タスクに関するもの」と「システム全体の状態を変えるもの」があります．

タスクに関するものは，優先順位の回転や現在実行中のタスクを知るためのものです．これらは，TSS（タイム・シェアリング・システム）のように処理をまんべんなく行うために優先順位を回転させたり，UNIXのinetd（インターネット・デーモン）のように，複数のリクエストに応えるため，同じ処理をタスクとして複数登録し，個々に自分のプロセスID（処理単位を判別するためのID，ITRONにおけるタスクIDのようなもの）を取得してふるまいを変えるようにするため用意されています．

システム全体の状態を変えるものは，割り込み禁止許可の状態や，CPUのロックなど，そのシステムに一つしかないものの状態を対象にしています．これらは，割り込み処理を記述するため，割り込みを禁止して処理したい場合などに，カーネルへ指示を与えるために使用します．特にこのサービス・コールは，システムに一つしかないものの状態を変えているため，処理の抜け，禁止後の許可のし忘れなどに注意する必要があります．

最後に，μITRON4.0仕様にはない項目「カーネル動作状態の参照」は，JSPカーネルの初期化が終了したかどうかを判断可能にします．これは機能拡張を行う際に必要となります．

● タスクの優先順位の回転

タスクの優先順位の回転については図1をご覧ください．実行可能状態にあるタスクのキュー（レディ・キュー）の優先順位の一番高いものを取り出し，優先度の一番低いところにつなぎます．

つないだ順序がそのままタスクの優先度となっているので，先ほどまで優先度の一番高かったタスクが，一番低くなります．

表1 TOPPERS/JSPカーネルのシステム管理機能

μITRON4.0仕様
タスクの優先順位の回転
実行状態のタスクIDの参照
CPUロック状態への移行
CPUロック状態の解除
ディスパッチの禁止
ディスパッチの許可
コンテキストの参照
CPUロック状態の参照
ディスパッチ禁止状態の参照
ディスパッチ保留状態の参照
TOPPERS/JSP
カーネル動作状態の参照

タスクの優先順位の回転

C言語API
```
ER rot_rdq(PRI tskpri);
ER irot_rdq(PRI tskpri);
```
パラメータ

PRI tskpri　　タスクの優先度

返り値

ER　E_OK　（正常終了）またはエラー・コード

エラー・コード

(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_PAR

E_CTX：コンテキスト・エラー
E_PAR：パラメータ・エラー

指定した優先度のタスクの優先順位を回転します．エラー・

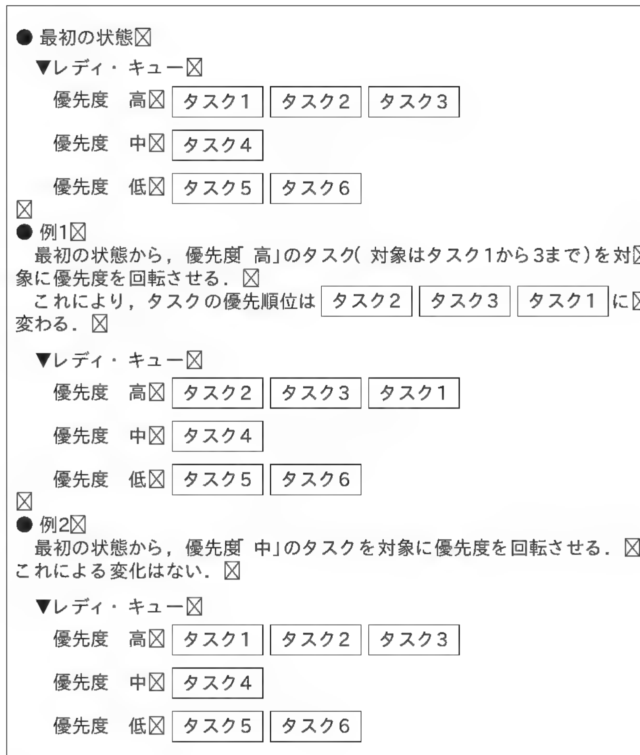


図1　タスクの優先順位の回転

コードに関しては，タスク内でirot_rdqを呼び出した場合や，逆にタスク外でrot_rdqを呼び出した場合に，E_CTXパラメータ・エラー，コンテキスト・エラーが返ります．指定した優先順位が誤っていた場合などはE_PARが返ります．

実行状態のタスクIDの参照

C言語API
ER get_tid(ID *p_tskid); ER iget_tid(ID *p_tskid);
パラメータ
ID *p_tskid　実行状態のタスクID
返り値
ER　E_OK　（正常終了）またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM)

現在実行中のタスクのIDを返します．実行中のタスクがない場合はTSK_NONE（0）を返します．

E_CTXコンテキスト・エラーが返るときは，呼び出しコンテキストを誤っているか，CPUロック状態にあります．

システム管理機能を使う場所

ここから先のシステム管理機能は，システム全体の状態を変えるものです．これらはタスクの切り替えを止めたり，タスクと非タスク間の切り替えを止めたりすることができます．

このような操作は，システム全体を止めてしまうため，注意して使用する必要があります．いつでも使って良いというわけではなく，どうしても必要な場面だけに限定して，使用するようにしなければなりません．

タスク間の切り替えや，タスク/非タスクの切り替えを止める処理は，どのような場面で使用するのでしょうか？　たとえば，アプリケーション・タスクが使用するアプリケーションの管理領域などを書き換える場面や，割り込み処理を入れ替える場面など，初期化にまつわる場面で使用することが多いようです．これらは一時的にシステム全体を止めてしまわないと，うまくいかない場合があったり，動作中に設定することによるコストが多大になるためです．

このコストとはどのようなものでしょうか？　動作中に再初期化するときに，変更した設定をシステム全体に行き渡らせるには，時間が必要です（図2）．そのため，一部だけ違う設定で動作する可能性があります．そこで同期を取るための時間や処理を行う必要が出てきます．このような場合，一度全部止めて設定を変え，再度システムを立ち上げなおすほうが速かったり，低コストになる場合もあります．

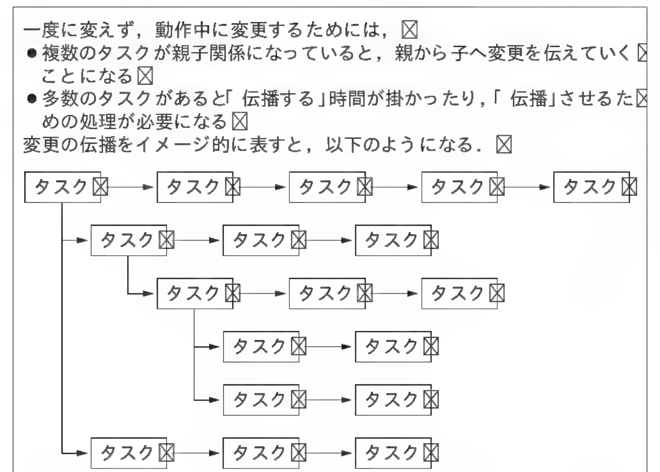


図2　変更した設定をシステム全体に行き渡らせる

CPUロック状態への移行

C言語API
ER loc_cpu(void); ER iloc_cpu(void);
パラメータ
なし
返り値
ER　E_OK　（正常終了）またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM)

コラム 1 コンテキストについて

プログラムの内部状態や置かれた状況，与えられた条件などをコンテキストといいます．たとえば，タスクの処理実行中であれば，処理の状況（タスク内で何処を実行しているか）を示すCPUのプログラム・カウンタや，そのほかのレジスタ群，タスクを管理するための変数などをいいます．

▶ タスク・コンテキスト

タスクの処理，タスクの処理の一部とみなせるコンテキストをタスク・コンテキストという

▶ 非タスク・コンテキスト

タスク・コンテキスト以外の総称

なお，CPU例外ハンドラは発生したコンテキストに依存しています．非タスク・コンテキストで発生した場合は非タスク・コンテキストとなるが，タスク・コンテキストで実行した場合に非タスク・コンテキストとなるかどうかは実装定義となっています．このため，実装によっては，どこで発生しても非タスク・コンテキストになるものがあったり，例外が発生した場所に依存するものがあります．このため，CPU例外ハンドラ内で使用できるサービス・コールはどちらのコンテキストか確認してから実行する必要があります．

ただし，TOPPERS/JSPでは，CPU例外ハンドラは一意に非タスク・コンテキストで実行するため，TOPPERS/JSPで作ったCPU例外ハンドラをほかのμITRON4.0の実装へ移植する，または逆にほかの実装からTOPPERS/JSPへ移植する場合には注意が必要です．

CPUロック状態の解除

C言語API
ER unl_cpu(void); ER iunl_cpu(void);
パラメータ
なし
返り値
ER E_OK　（正常終了）またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM)

CPUをロック状態にする，または解除します．E_CTXコンテキスト・エラーが返るときは，タスク外で使用するサービス・コールをタスク内で使用するなど，呼び出しコンテキストを誤っている場合です．

ディスパッチの禁止

C言語API
ER dis_dsp(void);
パラメータ
なし
返り値
ER E_OK　（正常終了）またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM)

ディスパッチを禁止します．ディスパッチとは，CPUがタスクを切り替えることを言います．TOPPERS/JSPカーネルの処理で，中心的な処理の一つです．

タスクを切り替えるタイミングは，サービス・コールを用いてカーネルに通知することになります．実際にはカーネルでの処理は極々短く簡単で，内部のパラメータを設定するだけなど，サービス・コール内ですべての処理が終わることもあります．

このディスパッチの禁止もTOPPERS/JSPカーネルでは，ごく軽い処理となっています．

E_CTXコンテキスト・エラーが返るときは，タスク外で使用するサービス・コールをタスク内で使用するなど，呼び出しコンテキストを誤っている場合です．

ディスパッチの許可

C言語API
ER ena_dsp(void);
パラメータ
なし
返り値
ER E_OK　（正常終了）またはエラー・コード
エラー・コード
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM)

ディスパッチを許可します．ディスパッチの禁止と同じく，E_CTXコンテキスト・エラーは，呼び出しコンテキストを誤っている場合に返ります．

コンテキストの参照

C言語API
BOOL sns_ctx(void);
パラメータ
なし
返り値
TRUEまたはFALSE

タスク・コンテキストか，非タスク・コンテキストかを参照します．戻り値は，非タスク・コンテキストの場合にTRUE，タスク・コンテキストの場合にFALSEを返します．タスク・コンテキストかそうでないかの判断基準は，「割り込み全許可でなければ非タスク・コンテキスト」としています．

CPUロック状態の参照

C言語API
BOOL sns_loc(void);
パラメータ
なし
返り値
TRUEまたはFALSE

CPUロック状態では，すべての割り込みが禁止されています．CPUロック状態でない場合は，CPUロック解除状態にあります．通常，タスクが動作中は，CPUロック解除状態にあります．

ディスパッチ禁止状態の参照

C言語API
BOOL sns_dsp(void);
パラメータ
なし
返り値
TRUEまたはFALSE

ディスパッチ禁止状態かディスパッチ許可状態かを参照します．ディスパッチ禁止状態の場合にTRUE，ディスパッチ許可状態の場合にFALSEを返します．

使用する場面としては，割り込み処理，例外処理などで，ディスパッチ禁止にする場合に，以前の状態を保存するといった目的のために使用します．

ディスパッチ保留状態の参照

C言語API
BOOL sns_dpn(void);
パラメータ
なし
返り値
TRUEまたはFALSE

ディスパッチ保留状態かそうでないかを参照します．

戻り値は，ディスパッチ保留状態の場合にTRUE，そうでない場合にはFALSEを返します．

TOPPERS/JSPカーネルには存在しませんが，システム内部に，ディスパッチャよりも優先順位が高い処理を用意した場合，その処理を実行中はディスパッチ保留状態とします．

カーネル動作状態の参照

C言語API
BOOL vsns_ini(void);
パラメータ
なし
返り値
TRUEまたはFALSE

μITRON仕様にはありませんが，実際のシステムを構築する場合に必要なJSPカーネル機能拡張を行う際には必要になります．

戻り値は，カーネルの初期化完了前または終了処理開始後に呼び出された場合にTRUEを返し，カーネルの動作中に呼び出された場合にFALSEを返します．

カーネル上で動作するタスクから呼び出される関数が，カーネルの初期化完了前や終了処理開始後にも呼び出される可能性がある場合には，その中でカーネルのサービス・コールを呼び出せるかを判別することが必要です．

たとえば，起動直後のハードウェア初期化処理など，いわゆる「ローダ」などで行い，次にTOPPERS/JSPカーネル上のアプリケーションに引き継ぐ場合などが該当します．これは一つの例ですが，起動時のメッセージをこのローダが行い，途中からTOPPERS/JSPカーネル上で動作するウィンドウ・システムなどに切り替える場合など，カーネルの初期化が完了したかどうかの確認が必要となります．これ以外にも，TOPPERS/JSPカーネル上のアプリケーションとカーネルの管理下にない処理から，カーネル上のアプリケーションに処理を行わせたい場合など，カーネルの初期化が完了しているかどうかを確認しておく必要がある場面で使用します．

割り込み管理機能

組み込みシステムはタスクだけでは成り立たず，どこかで外部のハードウェアとの接点があります．接点として，常時監視するポーリング処理もありますが，ある条件が成立したことを伝えるために割り込み処理を利用すると，システム全体をイベントで動かすことができます．

ハードウェアの監視をタスクに割りふって，ポーリングしながら処理要求があるかどうかを確認する場合，処理要求がなかった場合の処理時間がむだになります[注1]．

割り込み処理を利用した場合，このむだな時間が，ほぼなくなり，システム全体の処理が必要なときに必要な処理を行うイベント・ドリブンで行えるというメリットがあります．このため，ハードウェアからタスクへ処理要求のトリガを伝えるために割り込みが利用されます．

割り込みハンドラの定義（静的API）

静的API	
DEF_INH(INHNO inhno, { ATR inhatr, FP inthdr });	
パラメータ	
INHNO inhno	割り込みハンドラID
ATR inhatr	割り込みハンドラの属性
FP inthdr	割り込みハンドラにしたい関数への関数ポインタ

注1：ただし，割り込み処理にかかる時間，レジスタの保存や復帰のオーバヘッドが気になる場合は例外である．また，ハート・ビートを提供しているチック・タイムよりあまりにも周期が短い場合なども例外となる．むだが発生するとわかっていても，すばやく処理を行うために，ポーリング処理を行う場合があり，これらはわざと割り込み処理を使用していない．



2 DEF_INHの意味

DEF_INHで指定する割り込みハンドラ番号(inhno)について，おもなものを表Aに示します(TOPPERS/JSPのドキュメントのプロセッサ依存部に詳しい解説がある).

ほかのCPUに関しては，対応するボードにより複数の設定があるため，ここでは割愛します.

表A　割り込みハンドラ番号

Windowsエミュレータ	x86系のCPUではなく，68000 CPUでの例外ベクタ番号を元にしている
ARMv4	KS32C50100で周辺モジュールごとに定義されている割り込み番号を指定する．LH79532では割り込みコントローラの各チャネル番号を指定する
M68040	M68040での例外ベクタ番号を指定する
Microblaze	MHSファイルでデバイスに指定した割り込み優先度を指定する
SH-3	SH7708/50では割り込み事象レジスタ(INTEVT)に設定されるコードを指定する．SH7709A/09/29R/27では割り込み事象レジスタ2(INTEVT2)に設定されるコードを指定する
TMS320VC5402 DSP	C5402では，割り込みベクタ・ポインタ(IPTR)からの各割り込みベクタのオフセット値を右に2ビットだけシフト(4で除算)した値を指定する
Xstormy16	Xstormy16での割り込みベクタ番号を指定する．EVAデバッガのエミュレーション環境では，UART(ログ出力・標準入出力で使用)が16番に割り当てられいる

`INHNO`型の定義と`inhno`の意味はターゲットごとに定め，コンフィギュレーション・ファイルに用意します．これはCPU，サポートするボードごとに違います．割り込みハンドラの属性に関しては，TOPPERS/JSPカーネルでは`inhatr`に`TA_HLNG`のみを指定できます．

割り込みハンドラを定義したり，実際の処理を行うためには，以下のサービス・コールのいずれかのペアが必要です．この組み合わせは，禁止/許可のペアか，マスクの変更/参照のペアとなります．

以下のサービス・コールがサポートされているかどうか，サポートされている場合の仕様については，各CPUごとのターゲット依存となっています．具体的には，`chg_ixx`の`xx`の部分の名称や，型とパラメータの名称と意味，CPUロック状態やディスパッチ状態との関連などがターゲット依存部分となります．

この部分は，TOPPERS/JSPの各CPUのドキュメントを当たることになりますが，おもに以下のサービス・コールの名称で機能を提供しています．

● 割り込みの禁止

```
ER ercd = dis_int(INTNO intno);
```

● 割り込みの許可

```
ER ercd = ena_int(INTNO intno);
```

● 割り込みマスクの変更

```
ER ercd = chg_ixx(IXXXX ixxxx);
```

● 割り込みマスクの参照

```
ER ercd = get_ixx(IXXXX *p_ixxxx);
```

例外ハンドラ

TOPPERS/JSPカーネルでは，例外ハンドラの一つとして，CPU例外ハンドラをサポートしています．このCPU例外ハンドラは非タスク・コンテキストで実行され，この非タスク・コンテキストから呼び出せるサービス・コールは，CPU例外ハンドラ内からも呼び出すことができます．

実際に処理を行うには，どのような優先順位で実行されるかが重要になります．タスク・コンテキストを実行中にCPU例外が発生した場合，CPU例外ハンドラの優先順位はディスパッチャよりも高く，すべての割り込みハンドラおよびタイマ・ハンドラよりも低い設定になっています．これは，非タスク・コンテキストを実行中にCPU例外が発生した場合も同様で，CPU例外が発生した処理の優先順位よりも一つだけ高い優先順位となります．

最後に注意点として，CPU例外がCPUロック状態で発生した場合は，CPU例外ハンドラ中でCPUロックを解除することはできないため，非タスク・コンテキストで本来呼び出せるサービス・コールを呼び出すことができません．CPUロック状態にする処理がある場合は，細心の注意が必要です．

CPU例外ハンドラの定義(静的API)

静的API	
`DEF_EXC(EXCNO excno, { ATR excatr, FP exchdr });`	
パラメータ	
`EXCNO excno`	CPU例外ハンドラのID
`ATR excatr`	CPU例外ハンドラの属性
`FP exchdr`	CPU例外ハンドラにしたい関数への関数ポインタ

CPU例外ハンドラの定義を行います．`EXCNO`型の定義と`excno`の意味はターゲットごとに定める，または定まっています．コンフィギュレーション・ファイルには定めた値を指定します．これは割り込みハンドラと同様に，CPU，サポートするボードごとに違います．

コラム 3 ターゲット依存部分のサポート状況例示

ターゲット依存部分について，各プロセッサのサポート状況を表Bに示します．また，サポートしている場合のサービス・コールの名称を表Cに示します．

表C サポートしている場合のサービス・コール名称

def_inh	割り込みハンドラの定義
dis_int	割り込みの禁止
idis_int	割り込みの禁止
ena_int	割り込みの許可
iena_int	割り込みの許可
chg_ims	割り込みマスクの変更
ichg_ims	割り込みマスクの変更
get_ims	割り込みマスクの参照
iget_ims	割り込みマスクの参照

表B 各プロセッサのサポート状況

Windowsエミュレータ	割り込みハンドラ定義(def_int)，割り込みマスクの変更・参照(chg_ims，get_ims)，割り込みの禁止と許可(dis_int，ena_int)の五つをサポートしている
M68040	割り込みマスクの変更・参照(chg_ipm，get_ipm)をサポートしている．割り込みの禁止と許可(dis_int，ena_int)はサポートしていない．割り込みマスクの変更・参照機能は，SR(Status Register)中のIPM(Interrupt Priority Mask)の値を変更する
MIPS3	割り込みマスクの変更・参照(chg_ipm，get_ipm)をサポートしている．割り込みの禁止と許可(dis_int，ena_int)はサポートしていない
Microblaze	割り込みマスクの変更・参照(chg_ixx，get_ixx)，割り込みの禁止と許可(dis_int，ena_int)はサポートしていない
SH-1	割り込みマスクの変更・参照機能は，SR(Status Register)中の割り込みマスク・ビット(I3～I0)の値を変更する
SH-3	SR(Status Register)中の割り込みマスク・ビット(I3～I0)の値を変更する
H8	割り込みマスクの変更・参照(chg_ixx，get_ixx)，割り込みの禁止と許可(dis_int，ena_int)はサポートしていない
Linuxエミュレータ	名称はget_imsとしている．ref_imsは，シグナル・マスクの現在値を読み出すシステム・コールを用意している
Xstormy16	割り込みマスクの変更・参照(chg_ixx，get_ixx)，割り込みの禁止と許可(dis_int，ena_int)はサポートしていない．割り込みの許可・禁止およびレベル(優先度)はSFR(IL1L-IL2H)で管理している．これらの変更はSFRへの設定によって変更する

CPU例外ハンドラの属性に関しても，割り込みハンドラと同じく，TOPPERS/JSPカーネルではinhatrにTA_HLNGのみを指定できます．

CPU例外ハンドラについては，次のように記述してください．

```
void cpu_exception_handler(VP p_excinf)
    {
        CPU例外ハンドラ本体
    }
```

p_excinfには，CPU例外に関する情報を記憶している領域の先頭番地が渡されます．この情報を元に例外処理を行ってください．具体的には，CPU例外が発生したコンテキストや状態が情報として渡されます．

なお，CPU例外ハンドラからの復帰は，C言語の関数から単純にリターンすればよいことになっています．

● CPU例外ハンドラ内でできることとできないこと

μITRON4.0仕様では，CPU例外ハンドラ内で行えると規定されている操作は三つあります．TOPPERS/JSPでも同じように以下のことができます．これ以外のことはできません．

1) CPU例外が発生したコンテキストや状態の参照
TOPPERS/JSPカーネル独自のサービス・コール(vxsns_ctx，vxsns_loc，vxsns_dsp，vxsns_dpn，vxsns_tex)を用いて参照できる
2) CPU例外が発生したタスクのID番号の参照
iget_tidサービス・コールを用いて参照できる
3) タスク例外処理の要求
iras_texサービス・コールの呼び出しによって要求できる

そのほかの処理

ここから先は，ハードウェアに依存する部分と，ソフトウェア部品の再利用化を図るため，依存する部分，しない部分の切り口として，インターフェースが定義されています．

初期化ルーチンの追加(静的API)

静的API	
ATT_INI({ ATR iniatr, VP_INT exinf, FP inirtn });	
パラメータ	
ATR iniatr	初期化ルーチンの属性
VP_INT exinf	初期化ルーチンの拡張情報
FP inirtn	初期化ルーチンの起動番地(初期化ルーチンにしたい関数への関数ポインタ)

表2
実装独自のサービス・コール

CPU例外発生元のコンテキストの参照	BOOL vxsns_ctx(VP p_excinf);
CPU例外発生時のCPUロック状態の参照	BOOL vxsns_loc(VP p_excinf);
CPU例外発生時のディスパッチ禁止状態の参照	BOOL vxsns_dsp(VP p_excinf);
CPU例外発生時のディスパッチ保留状態の参照	BOOL vxsns_dpn(VP p_excinf);
CPU例外発生時のタスク例外処理禁止状態の参照	BOOL vxsns_tex(VP p_excinf);
性能評価用システム時刻参照機能	ER ercd = vxget_tim(SYSUTIM *p_sysutim);

コラム4 JSP以外の一覧

参考のため，TOPPERS/JSPでサポートしていないサービス・コールを表Dに示します．

これらはフルセットのカーネルでサポートしています．

表D TOPPERS/JSPでサポートしていないサービス・コール一覧

● 拡張同期・通信機能	
ミューテックス	
CRE_MTX	ミューテックスの生成(静的API)
cre_mtx	ミューテックスの生成
acre_mtx	ミューテックスの生成(ID番号自動割り付け)
del_mtx	ミューテックスの削除
loc_mtx	ミューテックスのロック
ploc_mtx	ミューテックスのロック(ポーリング)
tloc_mtx	ミューテックスのロック(タイム・アウトあり)
unl_mtx	ミューテックスのロック解除
ref_mtx	ミューテックスの状態参照
メッセージ・バッファ	
CRE_MBF	メッセージ・バッファの生成(静的API)
cre_mbf	メッセージ・バッファの生成
acre_mbf	メッセージ・バッファの生成(ID番号自動割り付け)
del_mbf	メッセージ・バッファの削除
snd_mbf	メッセージ・バッファへの送信
psnd_mbf	メッセージ・バッファへの送信(ポーリング)
tsnd_mbf	メッセージ・バッファへの送信(タイム・アウトあり)
rcv_mbf	メッセージ・バッファからの受信
prcv_mbf	メッセージ・バッファからの受信(ポーリング)
trcv_mbf	メッセージ・バッファからの受信(タイム・アウトあり)
ref_mbf	メッセージ・バッファの状態参照
ランデブ	
CRE_POR	ランデブ・ポートの生成(静的API)
cre_por	ランデブ・ポートの生成
acre_por	ランデブ・ポートの生成(ID番号自動割り付け)
del_por	ランデブ・ポートの削除
cal_por	ランデブの呼び出し
tcal_por	ランデブの呼び出し(タイム・アウトあり)
acp_por	ランデブの受け付け
pacp_por	ランデブの受け付け(ポーリング)
tacp_por	ランデブの受け付け(タイム・アウトあり)
fwd_por	ランデブの回送
rpl_rdv	ランデブの終了
ref_por	ランデブ・ポートの状態参照
ref_rdv	ランデブの状態参照

● メモリ・プール管理機能	
可変長メモリ・プール	
CRE_MPL	可変長メモリ・プールの生成(静的API)
cre_mpl	可変長メモリ・プールの生成
acre_mpl	可変長メモリ・プールの生成(ID番号自動割付け)
del_mpl	可変長メモリ・プールの削除
get_mpl	可変長メモリ・ブロックの獲得
pget_mpl	可変長メモリ・ブロックの獲得(ポーリング)
tget_mpl	可変長メモリ・ブロックの獲得(タイム・アウトあり)
rel_mpl	可変長メモリ・ブロックの返却
ref_mpl	可変長メモリ・プールの状態参照
● 時間管理機能	
アラーム・ハンドラ	
CRE_ALM	アラーム・ハンドラの生成(静的API)
cre_alm	アラーム・ハンドラの生成
acre_alm	アラーム・ハンドラの生成(ID番号自動割り付け)
del_alm	アラーム・ハンドラの削除
sta_alm	アラーム・ハンドラの動作開始
stp_alm	アラーム・ハンドラの動作停止
ref_alm	アラーム・ハンドラの状態参照
オーバラン・ハンドラ	
CRE_OVR	オーバラン・ハンドラの生成(静的API)
cre_ovr	オーバラン・ハンドラの生成
sta_ovr	オーバラン・ハンドラの動作開始
stp_ovr	オーバラン・ハンドラの動作停止
ref_ovr	オーバラン・ハンドラの状態参照
● そのほか	
システム状態管理機能	
ref_sys	システムの状態参照
サービス・コール管理機能	
DEF_SVC	拡張サービス・コールの定義(静的API)
def_svc	拡張サービス・コールの定義
cal_svc	サービス・コールの呼び出し
システム構成管理機能	
DEF_EXC	CPU例外ハンドラの定義(静的API)
def_exc	CPU例外ハンドラの定義
ref_cfg	コンフィギュレーション情報の参照
ref_ver	バージョン情報の参照
ATT_INI	初期化ルーチンの追加(静的API)

静的APIで初期化ルーチンの追加が行えます．初期化ルーチンの属性iniatrはTA_HLNGの指定はできますが，TA_ASM（初期化ルーチンをアセンブリ言語で記述する）をサポートしていません．

なお，登録された関数から例外が発生した場合の動作は未定義となっています．例外が発生しないことを確認しておく必要があります．

終了処理ルーチンの追加（静的API）

静的API	
VATT_TER({ ATR teratr, VP_INT exinf, FP terrtn });	
パラメータ	
ATR teratr	終了処理ルーチン属性
VP_INT exinf	終了処理ルーチンの拡張情報
FP terrtn	終了処理ルーチンの起動番地（終了処理ルーチンにしたい関数への関数ポインタ）

終了処理ルーチンの追加を行います．終了処理ルーチンの属性teratrはTA_HLNGの指定はできますが，TA_ASM（初期化ルーチンをアセンブリ言語で記述する）をサポートしていません．

静的API VATT_ATRによって追加された終了処理ルーチンは，システム終了処理時に実行されます．

実装独自サービス・コール

TOPPERS/JSPでは，CPU例外が発生したコンテキストや状態は，専用に用意されたサービス・コールで参照できるようになっています．このサービス・コールは，vxsns_ctx，vxsns_loc，vxsns_dsp，vxsns_dpn，vxsns_texの五つからなります．これらサービス・コールvxsns_yyyは，CPU例外が発生した処理では呼び出せないsns_yyyの代わりに利用できます．

これ以外にも，性能評価のためのサービス・コールも提供しています．表2を参照ください．

おわりに

最後のほうは，駆け足となりましたが，本連載のサービス・コールの解説は今回で終了です．次回以降は実際のCPUボードを使用した解説に移ります．解説記事執筆にあたり協力いただいたTOPPERSプロジェクトのメンバに感謝いたします．

最後に，筆者からのメッセージとして，もしあなたが新人ならば，ソースと仕様書を読んでみてください．量が多いですが，得るものも多いと思います．また，自分の腕試しのため，オープン・ソースへの貢献も考えてみてください．期待しています．

きしだ・まさみ　（株）フルノシステムズ





プログラミングの要

宮坂 電人

第14回 ソートとバイナリ・サーチ

ソート

ソートとは，データを大きいもの順（あるいは小さいもの順）に並べかえる操作をいいます．ソートには昔からいろいろなアルゴリズムが考案され，紹介されています．ここしばらく本連載のタネ本として使っている「Mastering Algorithms with C」[注1]でも何種類かソートのアルゴリズムを紹介しています．しかし実際にソートを実装する場合，できあいのライブラリを流用するか，もっとも実装が簡単そうなもの，またはもっとも実行速度が高速なものを利用することが多いでしょう．というわけで本連載ではもっとも実装が簡単と見られている「挿入ソート（Insertion Sort）」と，もっとも処理が高速だと見られている「クイック・ソート（Quick Sort）」の二つを取り上げます．

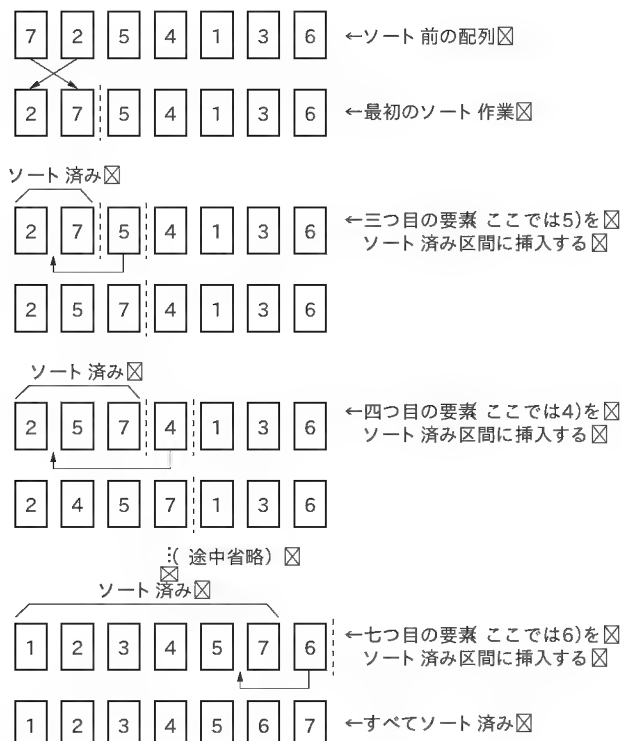


図1 挿入ソート

実装がもっとも単純なソートとして有名な方法に，バブル・ソートがあります．しかし，それと同時に実行速度が不利な点でも有名です．また，原理が単純で，だれでも簡単に思いつくようなアルゴリズムなので，わざわざ取り上げる必然性も少ないでしょう．というわけで，本連載ではバブル・ソートは取り上げません．

● 挿入ソート

その代わりとして紹介する挿入ソートは，実装が簡単である点と，あらかじめソートされている状況や，それに近い状況では高速に処理できるという点で有利です．

挿入ソートの原理は以下のようなものです（図1）．

1) 配列の最初の二つをソート済みにする
2) 三つ目の要素をすでにソート済みにした区間（この時点で2個の要素）に挿入する
3) 四つ目の要素をすでにソート済みにした区間（この時点で3個の要素）に挿入する

リスト1 挿入ソートの実装（InsSort.hpp）

```
template <typename T>
void InsSort(T* ioArray,int iArraySize)
{
    T aKey;
    for(int aJ = 1; aJ < iArraySize; aJ++){
        aKey = ioArray[aJ];
        int aI = aJ - 1;
        while(aI >= 0 && ioArray[aI] > aKey){
            ioArray[aI + 1] = ioArray[aI];
            --aI;
        }
        ioArray[aI + 1] = aKey;
    }
}
```

リスト2 InsSortの使用例

```
#define ARRAY_SIZE(X) (sizeof(X) / sizeof(X[0]))

static void demo()
{
    static int aData[] = { 24,52,11,94,28,36,14,80 };
    InsSort<int>(aData,ARRAY_SIZE(aData));
}
```

注1：http://www.oreilly.com/catalog/masteralgoc/を参照．

（途中省略）

n)（n＋1）つ目の要素をすでにソートずみにした区間（この時点で n 個の要素）に挿入する

という操作を繰り返していくと最終的に配列全体がソート済みになるという原理です．

この操作を C++ プログラムで実装するとリスト 1，リスト 2 のようになります．

● クイック・ソート

現時点でもっとも実行速度が高速なものと考えられているのがクイック・ソートです．クイック・ソートの原理は以下のようなものです（図 2）．

1) 集団の任意の一要素（これを pivot [注2] と称する）を選ぶ
2) 集団の全要素を走査し，pivot より小さい値を集めた集団と，pivot より大きい値を集めた集団の二つの集団を作り出す
3) 二つの集団の一つずつに対して 1)，2) の操作を再帰的に繰り返す

こうして作成された集団がすべて 1 要素になるまで上記の作業を繰り返し，多数できた集団を寄せ集めるとソート済みの集団ができるという原理です．

原理的には簡単なのですが，再帰を利用するという問題点（スタックあふれなど）や，すでにソート済みの配列をソートするときに pivot の決定が固定だと計算量が膨大になり，かえって遅くなるといった問題点などがあります．

後者に関しては pivot を固定にせずに乱数で決めることで問題を回避しやすくなります．「Mastering Algorithms with C」では pivot を決定するために，3 か所を乱数で求め，その三つの中央値を採用する手法にしていますが，本連載では単に 1 か所を乱数で決定する手法を採用しました．

実装はリスト 3 のようになります．一つの配列の内部で pivot の大小で二つの配列を作りだすためにはどうしたらよいのか悩むところですが，swap を利用し，pivot より小さいものが配列の先頭側に寄せられ，大きいものが末尾側に寄せられるようなプログラムにしています（図 3）．ある意味，パズルのようなプログラムですが，デバッガを使って 1 ステップずつ追跡すると，その動きがつかめると思います．

バイナリ・サーチ

本講座でハッシュ・テーブルやヒープなどの検索を高速に行う原理を紹介してきましたが，いずれも共通するのは検索する対象をすべて検索するのではなく「はしょっていく」ことで高速化することでした．

実は，単なる配列にデータを記録しても「はしょっていく」手法で高速検索できる可能性があります．ただし，前提として，

注2：日本語訳は「枢軸」，「軸要素」など．これ以外にもいろいろな日本語訳が付けられている．

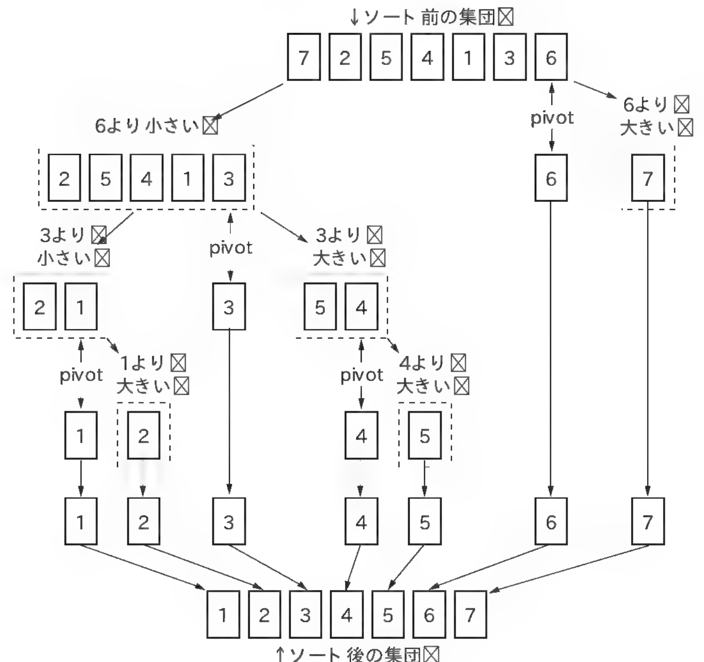


図 2　クイック・ソートの原理

リスト 3　クイック・ソートの実装（QuickSort.hpp）

```
#include <algorithm>
#include <cstdlib>

/* iFROM <= r <= iTOとなる整数の乱数をえる */
inline int RandomInt(int iFROM,int iTO)
{
    return iFROM + std::rand() % (iTO - iFROM + 1);
}

template <typename T>
int QS_Partition(T* ioArray,int iBegin,int iEnd)
{
    int aBegin = iBegin - 1;
    int aEnd = iEnd + 1;
    T aPval = ioArray[RandomInt(iBegin,iEnd)]; //pivotの決定
    for(;;){
        do{
            --aEnd;
        }while(aPval < ioArray[aEnd]);
        do{
            ++aBegin;
        }while(aPval > ioArray[aBegin]);
        if(aBegin >= aEnd){
            return aEnd;
        }
        std::swap<T>(ioArray[aBegin],ioArray[aEnd]);
    }
}

template <typename T>
void QuickSort(T* ioArray,int iBegin,int iEnd)
{
    if(iBegin < iEnd){
        int aPartition = QS_Partition<T>(ioArray,iBegin,iEnd);
        QuickSort<T>(ioArray,iBegin,aPartition);
        QuickSort<T>(ioArray,aPartition + 1,iEnd);
    }
}

template <typename T>
void QuickSort(T* ioArray,int iArraySize)
{
    QuickSort<T>(ioArray,0,iArraySize - 1);
}
```

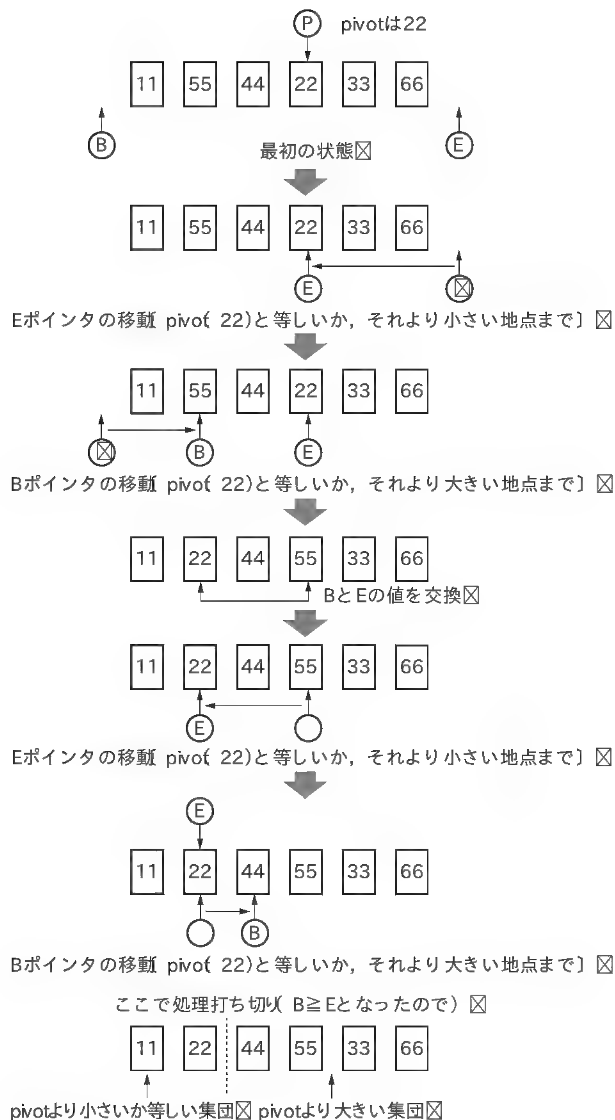


図3　配列上のクイック・ソートの動き

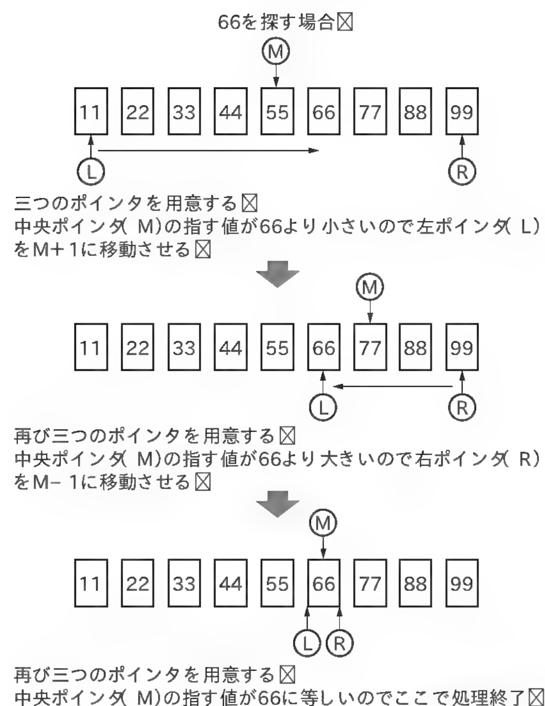


図4　バイナリ・サーチ

リスト4　バイナリ・サーチの実装（BinarySearch.hpp）

```
template <typename T>
int BinarySearch(const T* iArray,int iArraySize,const T& iTarget)
{
    int aLeft = 0;
    int aRight = iArraySize - 1;
    while(aLeft <= aRight){
        int aMiddle = (aLeft + aRight) / 2;
        const T& aMidObj = iArray[aMiddle];
        if(aMidObj == iTarget){
            return aMiddle;
        }
        if(aMidObj < iTarget){
            aLeft = aMiddle + 1;
        }else{
            aRight = aMiddle - 1;
        }
    }
    return -1;
}
```

リスト5　バイナリ・サーチの使用例

```
static void bs_test(int iFind)
{
    static int aData[] = { 11,14,24,28,36,52,80,94 };
    int aPos;
    aPos = BinarySearch<int>(aData,ARRAY_SIZE(aData),iFind);
    std::cout << "pos:" << iFind << " = " << aPos << std::endl;
}
```

あらかじめ配列の内容がソート済みであることが条件です．その前提で以下のような検索を行うのがバイナリ・サーチと呼ばれる手法です（図4）．

1）配列の左端を指すポインタ（L）と右端を指すポインタ（R）を用意する
2）両ポインタの中間地点を指すポインタ（M）を用意する
3）ポインタ（M）が指す値が目的の値であるなら，ここで終了する
4）ポインタ（M）が指す値が目的の値より低いなら，ポインタ（L）をM＋1とし，6）に行く
5）さもなくば，ポインタ（R）をM－1とし，6）に行く
6）ポインタ（R）＜ポインタ（L）なら見つからなかったので，ここで終了する．さもなくば2）に行く

三つ用意したポインタを上手に駆使して「はしょっていく」ことで高速に検索できるわけです．

バイナリ・サーチは「二分探索法」ともいいます．これに対して従来どおり配列を先頭から末尾まで一つ一つ検索する方法を「リニア・サーチ」または「線形探索法」と呼ぶことがあります．

バイナリ・サーチをC++プログラムで実装したのがリスト4，リスト5です．

表1　qsort(stdlib.h)

記述	void qsort(void *base,size_t nmemb, size_t size,int (*compare)(const void *, const void *));	
引き数	base	ソートしたい配列
	nmemb	配列の全要素数
	size	配列の一要素のサイズ
	compare	要素どうしの比較関数
返り値	(なし)	
説明	指定した配列をソートする	

表2　sort(algorithm)

記述	template<class RanIter> void sort(RanIter RanIter end); または， template<class RanIter,class Comp> void sort (RanIter start,RanIter end,Comp cmpfn);	
引き数	RanIter	ランダム・アクセス可能なイテレータを示す型
	Comp	比較関数を示す型
	start	ソート対象の開始地点
	end	ソート対象の終了地点+1
	cmpfn	比較関数へのポインタ，あるいは関数オブジェクト
返り値	(なし)	
説明	イテレータで指定した範囲をソートする	

表3　sort(list)

記述	void sort(); または， template <class Comp> void sort(Comp cmpfn);	
引き数	Comp	比較関数を示す型
	cmpfn	比較関数へのポインタ，あるいは関数オブジェクト
返り値	(なし)	
説明	listの全内容をソートする	

表4　bsearch(stdlib.h)

記述	void *bsearch(const void *key,const void *base,size_t num,size_t size,int (*compare)(const void *,const void *));	
引き数	key	検索したい値
	base	検索対象の配列
	num	配列の全要素数
	size	配列の一要素のサイズ
	compare	要素同士の比較関数
返り値	発見した場所，発見できなかったならNULLを返す	
説明	指定した配列を検索する	

標準ライブラリのソート

今回，ソートとバイナリ・サーチのアルゴリズムを紹介しましたが，実際にC言語やC++でプログラムする場合，アルゴリズムを元に実装する例は極めてまれでしょう．

というのも，すでに標準関数やSTLに用意されているので，それらを利用するほうが楽だからです．

ソートはC言語の標準関数だとqsort(表1)が使えます．

qsortは有名な標準関数なのですが，比較関数の指定が理解されていないため，案外使われていないという悲しい状況もあるようです．比較関数の引き数には配列の各要素へのポインタが指定されます．そして，

第1引き数の指す値<第2引き数の指す値ならマイナス

第1引き数の指す値==第2引き数の指す値ならゼロ

第1引き数の指す値>第2引き数の指す値ならプラス

を返すように作成しておきます(リスト6)．

リスト6　qsortの使用例

```
static int IntComp(const void *i1,const void *i2)
{
    const int *a1 = (const int *)i1;
    const int *a2 = (const int *)i2;
    return *a1 - *a2;
}

static void demo()
{
    static int aData[] = { 24,52,11,94,28,36,14,80 };
    qsort(aData,ARRAY_SIZE(aData),sizeof(int),IntComp);
}
```

リスト7 sortの使用例

```
static void demo()
{
    static int aData[] = { 24,52,11,94,28,36,14,80 };
    std::vector<int> aV;
    aV.assign(aData,aData + ARRAY_SIZE(aData));
    std::sort(aV.begin(),aV.end(),std::greater<int>());
}
```

C++だとランダム・アクセス可能なイテレータで指す領域をソートするsortというアルゴリズム[注3]が使えます(リスト7, 表2)．ランダム・アクセス可能なイテレータが使えないlist(双方向連結リストをSTLで実装したテンプレート)でもソートのメンバ関数が用意されています(表3)．

標準ライブラリのバイナリ・サーチ

C言語の標準関数では，バイナリ・サーチとしてbsearchが使えます(表4)．

bsearchも有名な標準関数なのですが，qsortと同様，比較関数の指定が理解されていないため案外使われない悲しい状況があるようです(リスト8)．

C++だと双方向イテレータ[注4]で指す領域から検索を行うbinary_searchというアルゴリズム(表5)が使えます(リスト9)．

本講座で紹介したデータ構造とSTL

今まで本講座で紹介したデータ構造についても，C++とSTL

注3：ここで称しているアルゴリズムとは，STLの機能分類としての名称であるのに注意．

注4：STLでの表記は「bidirectional iterator」．値の設定と取得ができ，移動は前進と後進の両方ができるイテレータのこと．

でプログラミングできる環境であるなら，すでに用意されているテンプレート・ライブラリを利用できる可能性があります．利用するにあたって若干ややこしい約束事があったり，テンプレート特有の弊害[注5]が予想されますが，なんといっても標準ライブラリであるという安心感があります．

リスト 8　bsearchの使用例

```
static void bs_test(int iFind)
{
    static int aData[] = { 11,14,24,28,36,52,80,94 };
    const int *aPtr = (const int *)bsearch(
        &iFind,aData,ARRAY_SIZE(aData),sizeof(int),IntComp);
    if(aPtr == NULL){
        printf("not found (%d)¥n",iFind);
    }else{
        printf("found (%d) at pos:%d¥n",iFind,
                                    (aPtr - aData));
    }
}
```

リスト 9　binary_searchの使用例

```
static void bs_test(int iFind)
{
    static int aData[] = { 11,14,24,28,36,52,80,94 };
    std::list<int> aList;
    aList.assign(aData,aData + ARRAY_SIZE(aData));
    aList.sort();
    if(std::binary_search(aList.begin(),aList.end(),iFind)){
        std::cout << "found (" << iFind << ")¥n";
    }else{
        std::cout << "not found (" << iFind << ")¥n
    }
}
```

● 双方向連結リスト（第11回）

STLで双方向連結リストを実現したい場合，`list`を使います．`list`はたいていの実装では，

```
template <class T,class Allocator =
                    allocator<T> > class list
```

となっています．`T`は任意の型を選べます．`Allocator`はオブジェクトを確保するときのアロケータの指定ですが，通常はデフォルト設定を使い，明示的に変更する機会は少ないでしょう．`list`にはメンバ関数が多数ありますが，おもなものは表6のとおりです（リスト10）．

表5　binary_search（algorithm）

記述	template <class ForIter,class T> bool binary_search(ForIter start,ForIter end, const T& val); または， template <class ForIter,class T,class Comp> bool binary_search(ForIter start,ForIter end, const T& val,Comp cmpfn);	
引き数	ForIter	双方向イテレータを示す型
	T	任意の型
	Comp	比較関数を示す型
	start	検索対象の開始地点
	end	検索対象の終了地点+1
	val	検索したい値
	cmpfn	比較関数へのポインタ，あるいは関数オブジェクト
返り値	発見したならtrue，そうでないならfalse	
説明	検索対象を検索する．あらかじめ検索対象はソート済みであるという前提	

表6　list

記述	iterator insert(iterator i,const T& val = T());	
引き数	i	挿入地点の指定
	val	挿入するオブジェクト
返り値	挿入したオブジェクトを指すイテレータ	
説明	iで示すオブジェクトの直前にvalを挿入する	

（a）insert（list）

記述	iterator erase(iterator i);	
引き数	i	削除する地点の指定
返り値	削除位置の次を指すイテレータ	
説明	iで示すオブジェクトを削除する	

（b）erase（list）

記述	size_type size() const;
引き数	（なし）
返り値	listに記録されている全オブジェクト数
説明	listに記録されている全オブジェクト数を返す

（c）size（list）

記述	reference front(); または， const_reference front() const;
引き数	（なし）
返り値	listの先頭へのリファレンス
説明	listの先頭へのリファレンスを返す

（d）front（list）

記述	iterator begin(); または， const_iterator begin() const;
引き数	（なし）
返り値	listの先頭を指すイテレータ
説明	listの先頭を指すイテレータを返す

（e）begin（list）

記述	reference back(); または， const_reference back() const;
引き数	（なし）
返り値	listの末尾へのリファレンス
説明	listの末尾へのリファレンスを返す

（f）back（list）

記述	iterator end(); または， const_iterator end() const;
引き数	（なし）
返り値	listの末尾+1を指すイテレータ
説明	listの末尾+1を指すイテレータを返す

（g）end（list）

注5：この点に関しては本誌2004年2月号特集記事が参考になる．

● スタック(第11回)

STLでスタックを実現したい場合，stackを使います．stackはたいていの実装では，

```
template <class T,class Container =
                         deque<T> > class stack
```

となっています．Tは任意の型を選べます．

Containerは実際にスタックを実装するコンテナ[注6]を意味します．実はstackと，この後で説明するqueueはSTLではコンテナ・アダプタという範ちゅうのテンプレート・ライブラリです．コンテナ・アダプタは別のコンテナをベースにしてコンテナを作成するアプローチです[注7]．特に必然性がないならデフォルトのdequeを指定しておくとよいでしょう．stackのおもなメンバ関数は表7のとおりです(リスト11)．

● キュー(第11回)

STLでキューを実現したい場合，queueを使います．queueはたいていの実装では，

```
template <class T,class Container =
                         deque<T> > class queue
```

となっています．Tは任意の型を選べます．queueのおもなメンバ関数は表8のとおりです(リスト12)．

リスト10　listの使用例

```
int aI,aJ;
std::list<int> aList;
for(aI = 0; aI < 10; aI++){
    aList.insert(aList.end(),aI);
}
for(aJ = 0; aJ < 3; aJ++){
    aI = aList.size();
    std::cout << "aI = " << aI << std::endl;
    aI = aList.front();
    std::cout << "aI = " << aI << std::endl;
    aI = aList.back();
    std::cout << "aI = " << aI << std::endl;
    aList.erase(aList.begin());
}
```

リスト11　stackの使用例

```
int aI,aJ;
std::stack<int> aStack;
for(aI = 0; aI < 9; aI++){
    aStack.push(aI);
}
aI = aStack.size();
std::cout << "aI = " << aI << std::endl;
for(aJ = 0; aJ < 3; aJ++){
    aI = aStack.top();
    std::cout << "aI = " << aI << std::endl;
    aStack.pop();
}
```

リスト12　queueの使用例

```
int aI,aJ;
std::queue<int> aQueue;
for(aI = 0; aI < 9; aI++){
    aQueue.push(aI);
}
aI = aQueue.size();
std::cout << "aI = " << aI << std::endl;
for(aJ = 0; aJ < 3; aJ++){
    aI = aQueue.front();
    std::cout << "aI = " << aI << std::endl;
    aQueue.pop();
}
```

表7　stack

記述	size_type size() const;	
引き数	(なし)	
返り値	stackに記録されている全オブジェクト数	
説明	stackに記録されている全オブジェクト数を返す	

(a) size(stack)

記述	void push(const T& val);	
引き数	val	格納するオブジェクト
返り値	(なし)	
説明	stackにオブジェクトをプッシュする	

(b) push(stack)

記述	value_type& top(); または， const value_type& top() const;
引き数	(なし)
返り値	stackのトップにあるオブジェクトへのリファレンス
説明	stackのトップにあるオブジェクトへのリファレンスを返す

(c) top(stack)

記述	void pop();
引き数	(なし)
返り値	(なし)
説明	stackのトップにあるオブジェクトを削除する

(d) pop(stack)

表8　queue

記述	size_type size() const;
引き数	(なし)
返り値	queueに記録されている全オブジェクト数
説明	queueに記録されている全オブジェクト数を返す

(a) size(queue)

記述	void push(const T& val);	
引き数	val	格納するオブジェクト
返り値	(なし)	
説明	queueの末尾にオブジェクトを登録する	

(b) push(queue)

記述	value_type& front(); または， const value_type& front() const;
引き数	(なし)
返り値	queueの先頭にあるオブジェクトへのリファレンス
説明	queueの先頭にあるオブジェクトへのリファレンスを返す

(c) front(queue)

記述	void pop();
引き数	(なし)
返り値	(なし)
説明	queueの先頭にあるオブジェクトを削除する

(d) pop(queue)

注6：複数のオブジェクトを登録できるオブジェクトのこと．配列や集合の特性を実装すればコンテナとなる．STLではvector，deque，listが代表的なコンテナ．
注7：文字どおり，デザイン・パターンでいうところのAdapterパターンの適用例である．

表9　set

記述	size_type size() const;
引き数	(なし)
返り値	set に記録されている全オブジェクト数
説明	set に記録されている全オブジェクト数を返す

(a) size(set)

記述	iterator begin(); または, const_iterator begin() const;
引き数	(なし)
返り値	set の先頭を指すイテレータ
説明	set の先頭を指すイテレータを返す

(b) begin(set)

記述	iterator end(); または, const_iterator end() const;
引き数	(なし)
返り値	set の末尾+1を指すイテレータ
説明	set の末尾+1を指すイテレータを返す

(c) end(set)

記述	pair<iterator,bool> insert(const value_type& val);	
引き数	val	挿入するオブジェクト
返り値	挿入結果．挿入できたなら pair<iterator,true> を返し，できなければ pair<iterator,false> を返す．できたときは iterator は挿入したオブジェクトを指している	
説明	set にオブジェクトを挿入する	

(d) insert(set)

記述	iterator find(const key_type& k) const;	
引き数	k	検索したいオブジェクト
返り値	検索できたなら，そのオブジェクトへのイテレータを返し，できなかったなら end() を返す	
説明	set に登録されているオブジェクトを検索する	

(e) find(set)

記述	size_type erase(const key_type& k);	
引き数	k	削除したいオブジェクト
返り値	削除されたオブジェクトの個数	
説明	set に登録されているオブジェクトを削除する	

(f) erase(set)

リスト13　setの使用例

```
static void setupIntSets(std::set<int>& oSets,int i1,int i2)
{
    for(int aI = i1; aI <= i2; aI++){
        oSets.insert(aI);
    }
}

static void findSetTest(const std::set<int>& iSets,int iFind)
{
    std::set<int>::iterator aSetItr = iSets.find(iFind);
    if(aSetItr == iSets.end()){
        std::cout << "not found : " << iFind << std::endl;
    }else{
        std::cout << "found : " << iFind << std::endl;
    }
}

static void eraseSetTest(std::set<int>& iSets,int iErase)
{
    int aResult = iSets.erase(iErase);
    std::cout << "erase : " << iErase << " -> " << aResult
                                                  << std::endl;
}

static void demo1()
{
    std::set<int> aSet1;
    setupIntSets(aSet1,1,5);
    findSetTest(aSet1,3);
    findSetTest(aSet1,-3);
    eraseSetTest(aSet1,3);
    eraseSetTest(aSet1,-3);
}
```

表10　includes(algorithm)

記述	template <class InIter1,class InIter2> bool includes(InIter1 start1,InIter1 end1, InIter2 start2,InIter2 end2); または, template <class InIter1,class InIter2,Comp cmpfn> bool includes(InIter1 start1, InIter1 end1,InIter2 start2,InIter2 end2, Comp cmpfn);	
引き数	InIter1,InIter2	入力イテレータの型
	start1,end1	集合1
	start2,end2	集合2
	Comp	比較関数を示す型
	cmpfn	比較関数へのポインタ，あるいは関数オブジェクト
返り値	判定結果	
説明	集合1が集合2の全要素を含むなら true を返し，そうでないなら false を返す	

●集合(第12回)

STLで集合を扱う場合，set，multiset，map，multimapあたりを利用します．ここではもっとも扱いやすいsetについて説明します．setは，たいていの実装では，

```
template <class Key,class Comp = less<Key>,
class Allocator = allocator<Key> > class set
```

となっています．Keyは任意の型を選べますが，なんらかの順序でソートされる基準があるという前提です．Compはソートのときに利用する比較関数を意味し，デフォルトでは，less(「bool c = std::less<int>()(a,b);」というコードがあったとき，a＜bならtrue，a≧bならfalseがcに入る関数オブジェクト)が指定されます．setにはメンバ関数が多数ありますが，おもなものは表9のとおりです(リスト13)．

STLでは，集合に関する操作アルゴリズムがいくつか用意されています(表10～表13)．ただし，操作はsetだけに限定されておらず，ソート済みのシーケンス・コンテナ[注8]に対しても適用できるのが便利です．

注8：STLではオブジェクトを連続的に格納しているコンテナをシーケンス・コンテナと称する．具体的にはvector，deque，listなど．

表11　set_union（algorithm）

記述	template <class InIter1,class InIter2, class OutIter> OutIter set_union(InIter1 start1,InIter1 end1,InIter2 start2,InIter2 end2,OutIter result); または， template <class InIter1,class InIter2, class OutIter,Comp cmpfn> OutIter set_union(InIter1 start1,InIter1 end1, InIter2 start2,InIter2 end2, OutIter result,Comp cmpfn);	
引き数	InIter1,InIter2	入力イテレータの型
	OutIter	出力イテレータの型
	start1,end1	集合1
	start2,end2	集合2
	result	処理結果を入れる場所
	Comp	比較関数を示す型
	cmpfn	比較関数へのポインタ， あるいは関数オブジェクト
返り値	処理結果の末尾を示すイテレータ	
説明	集合1と集合2の和集合（集合1の要素と集合2の要素の両方を含むもの）を作成しresultに格納する	

表12　set_difference（algorithm）

記述	template <class InIter1,class InIter2, class OutIter> OutIter set_difference (InIter1 start1,InIter1 end1,InIter2 start2,InIter2 end2,OutIter result); または， template <class InIter1,class InIter2, class OutIter,Comp cmpfn> OutIter set_difference(InIter1 start1,InIter1 end1,InIter2 start2,InIter2 end2,OutIter result,Comp cmpfn);	
引き数	InIter1,InIter2	入力イテレータの型
	OutIter	出力イテレータの型
	start1,end1	集合1
	start2,end2	集合2
	result	処理結果を入れる場所
	Comp	比較関数を示す型
	cmpfn	比較関数へのポインタ， あるいは関数オブジェクト
返り値	処理結果の末尾を示すイテレータ	
説明	集合1と集合2の差集合（集合1の要素から集合2にある要素を削除したもの）を作成しresultに格納する	

表13　set_intersection（algorithm）

記述	template <class InIter1,class InIter2, class OutIter> OutIter set_intersection (InIter1 start1,InIter1 end1,InIter2 start2,InIter2 end2,OutIter result); または， template <class InIter1,class InIter2, class OutIter,Comp cmpfn> OutIter set_intersection(InIter1 start1,InIter1 end1, InIter2 start2,InIter2 end2,OutIter result,Comp cmpfn);	
引き数	InIter1,InIter2	入力イテレータの型
	OutIter	出力イテレータの型
	start1,end1	集合1
	start2,end2	集合2
	result	処理結果を入れる場所
	Comp	比較関数を示す型
	cmpfn	比較関数へのポインタ， あるいは関数オブジェクト
返り値	処理結果の末尾を示すイテレータ	
説明	集合1と集合2の積集合（集合1と集合2の両方にある要素を取り出したもの）を作成しresultに格納する	

集合を作成するアルゴリズムで注意しなければならない点は，作成された集合はresultで示した場所から上書きされて作成されるため，挿入で作成させたいのなら上書きではなくinsert_iteratorのようなイテレータ・アダプタを併用する必要があるという点です．これはSTLを使う人がハマってしまう落とし穴の一つです．

● ハッシュ・テーブルとヒープ（第12回）

C++コンパイラによってはhash_set，hash_mapというハッシュ・テーブルのテンプレートを用意している例もありますが，残念ながら今のところ標準ライブラリ扱いではないため，説明は省略します．

また，ヒープについてはSTLのアルゴリズムでmake_heap，push_heap，pop_heapが用意されていますが，実際に使われている例を見たことがありません．そもそも，ヒープを利用する最大の動機ともいえる優先順位付きキューがpriority_queueというコンテナ・アダプタとして用意されているので，そちらを使うほうがてっとり早いかと思われます．

みやさか・でんと　miyadent@anet.ne.jp

やり直しのための信号数学 第26回（最終回）

総まとめⅢ（DFT，DCT編）

三谷 政昭



前回は，DFT値（スペクトル）が表す物理的な意味，高速計算アルゴリズムのFFTについて説明した．とくに，FFTの考え方の土台がDFT計算における“共通計算の抽出”にあることを示した．

今回は「DFT，DCT編」と題して，まずIDFT（逆ディジタル・フーリエ変換）が連立方程式の解に相当することを示した後，DFTの諸性質，相関関数とDFTの相互関係について概括する．次に，画像に対するディジタル・コサイン変換（2次元DCT）を採り上げ，正規直交基底ベクトルによる画像表現に基づき，2次元DCT値の物理的な意味を中心に解説する．最後に，DFTとDCTの着目すべき適用ポイントをまとめる．（筆者）

IDFTは連立方程式（DFT計算式）の解

まずは，DFTの逆変換である“逆ディジタル・フーリエ変換（以後，IDFTと略記）”が連立方程式の解に相当することを，4サンプルのディジタル信号を例に実感してもらおう．

4サンプルに対するDFT計算式は，

$$\begin{cases} X_0=\frac{1}{4}\left[x_0+x_1+x_2+x_3\right] & \cdots(1) \\ X_1=\frac{1}{4}\left[x_0-jx_1-x_2+jx_3\right] & \cdots(2) \\ X_2=\frac{1}{4}\left[x_0-x_1+x_2-x_3\right] & \cdots(3) \\ X_3=\frac{1}{4}\left[x_0+jx_1-x_2-jx_3\right] & \cdots(4) \end{cases}$$

で与えられる．ここで，式(1)～式(4)を4個の未知変数$\{x_0, x_1, x_2, x_3\}$に関する連立方程式とみなして，方程式の解を効率的に算出する手順がIDFTの計算に相当することを示す．なお，中学校で学んだ連立方程式の解法としては，一般に「代入法」や「加減法」が知られているが，ここでは「変数一括消去法」とでもネーミングできる“超”簡単な手法を紹介する．

手始めに，x_1を算出する手順を示そう．考え方は，[ ]内のx_1の各係数の複素共役を両辺に乗じ，総和を採るのである．

たとえば，式(4)のx_1の係数は$j(=\sqrt{-1})$なので，複素共役は$(-j)$となる．この$(-j)$を両辺に乗じて，$j^2=-1$を適用することにより，

$$-jX_3=\frac{1}{4}\left[-jx_0-j^2x_1+jx_2+j^2x_3\right]$$

$$=\frac{1}{4}\left[-jx_0+x_1+jx_2-x_3\right] \cdots(5)$$

と表される関係が得られる．

ほかも同様で，式(1)～式(3)におけるの係数の複素共役がそれぞれ，1，j，(-1)なので，次のように計算される．

$$X_0=\frac{1}{4}\left[x_0+x_1+x_2+x_3\right] \cdots(6)$$

$$jX_1=\frac{1}{4}\left[jx_0-j^2x_1-jx_2+j^2x_3\right]$$

$$=\frac{1}{4}\left[jx_0+x_1-jx_2-x_3\right] \cdots(7)$$

$$-X_2=\frac{1}{4}\left[-x_0+x_1-x_2+x_3\right] \cdots(8)$$

以上より，式(5)～式(8)の右辺と左辺を別々に加算してみると，あっと驚かれるであろう．すなわち，x_0，x_2，x_3に関する係数をすべて消去する(0にする)ことができて，

$$X_0+jX_1-X_2-jX_3$$

$$=\frac{1}{4}\underbrace{\{1+j-1-j\}}_{0}x_0+\frac{1}{4}\underbrace{\{1+1+1+1\}}_{4}x_1+\frac{1}{4}\underbrace{\{1-j-1+j\}}_{0}x_2+\frac{1}{4}\underbrace{\{1-1+1-1\}}_{0}x_3$$

$$=x_1$$

となり，いとも簡単に未知変数x_1の値が得られるのである．ほかの未知変数に対しても同様な計算により連立方程式の解を算出することができ，



$$\begin{cases} x_0 = X_0 + X_1 + X_2 + X_3 \\ x_1 = X_0 + jX_1 - X_2 - jX_3 \\ x_2 = X_0 - X_1 + X_2 - X_3 \\ x_3 = X_0 - jX_1 - X_2 + jX_3 \end{cases} \quad \cdots\cdots (9)$$

となる関係が得られる（各自で検証してもらいたい）．驚くことなかれ，式（9）で表される関係がまさしく逆ディジタル・フーリエ変換（IDFT）なのである．なお，サンプル数 N［個］のディジタル信号 $\{x_k\}_{k=0}^{k=N-1}$，DFT 値 $\{X_\ell\}_{\ell=0}^{\ell=N-1}$ の場合の IDFT 計算の一般式を以下に示しておく．

$$x_k = \sum_{\ell=0}^{N-1} X_\ell W_N^{-k\ell} \quad \cdots\cdots (10)$$

ただし，$W_N = e^{-j\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) - j\sin\left(\frac{2\pi}{N}\right)$

DFTのいろいろな性質

いま，信号 $\{x_k\}_{k=0}^{k=N-1}$，$\{y_k\}_{k=0}^{k=N-1}$ に対する DFT 値をそれぞれ，$\{X_\ell\}_{\ell=0}^{\ell=N-1}$，$\{Y_\ell\}_{\ell=0}^{\ell=N-1}$ と表すことにし，DFT の諸性質を以下にまとめておく．

（i）周期性

図 26.1（a）の周期信号の基本区間 $\{x_k\}_{k=0}^{k=N-1}$ に対する DFT 値 $\{X_\ell\}_{\ell=0}^{\ell=N-1}$ は，

$$X_\ell = \frac{1}{N}\sum_{k=0}^{N-1} x_k W_N^{k\ell} \quad \cdots\cdots (11)$$

と表される．このとき，r を任意の整数として，図 26.1（b）に示す基本区間の繰り返し波形である周期信号は，

$$\{x_{k+rN}\}_{k=0}^{k=N-1} \quad \cdots\cdots (12)$$

と表され，DFT 値 $\{\tilde{X}_\ell\}_{\ell=0}^{\ell=N-1}$ は，

$$\begin{aligned} \tilde{X}_\ell &= \frac{1}{N}\sum_{k=0}^{N-1} x_{k+rN} W_N^{(k+rN)\ell} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} x_k W_N^{k\ell} \times W_N^{rN\ell} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} x_k W_N^{k\ell} \times (\underbrace{W_N^{\,N}}_{1})^{r\ell} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} x_k W_N^{k\ell} = X_\ell \end{aligned} \quad \cdots\cdots (13)$$

となり，時間波形と同様に周期性を有することがわかる［図 26.1（c）］．

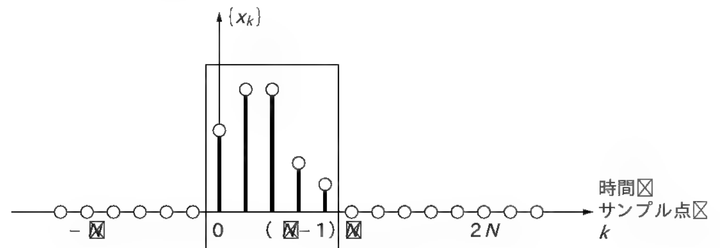
（ii）線形性

いま，a と b を実数の定数として，信号 x_k の a 倍と信号 y_k の b 倍したものを合成した信号，すなわち，

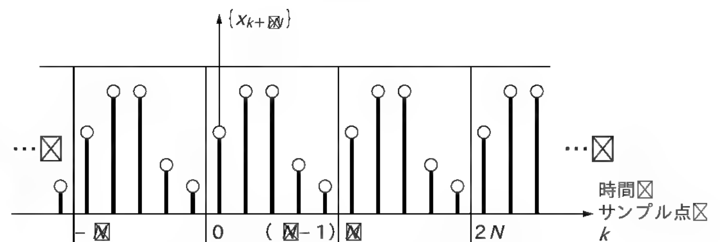
$$\{ax_k + by_k\}_{k=0}^{k=N-1} \quad \cdots\cdots (14)$$

に対する DFT 値は，

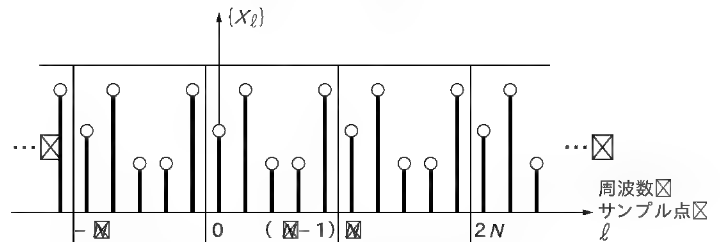
$$\{aX_\ell + bY_\ell\}_{\ell=0}^{\ell=N-1} \quad \cdots\cdots (15)$$



（a）基本区間



（b）周期波形



（c）周期スペクトル

図 26.1　周期性（N＝5の場合）

となる．つまり，信号が n 倍になれば，それにともなって DFT 値（周波数成分）も n 倍になるということを意味する．

（iii）時間シフト

まず，時間シフト（時間推移ともいう）について説明する．図 26.2（a）に示すディジタル信号 $\{x_{k+rN}\}_{k=0}^{k=N-1}$ を m サンプルだけ右へシフトする（ずらす）と，図 26.2（b）となり，

$$\{x_{k-m+rN}\}_{k=0}^{k=N-1} \quad \cdots\cdots (16)$$

と表される．このとき，添え字（$k-m$）は，

$$(k-m) \rightarrow (k-m) \mod N \quad \cdots\cdots (17)$$

のように，サンプル数 N［個］で割り算した余りとして与えるので，最終的には図 26.2（c）が m サンプルだけ右へ時間シフトした波形となる．なお，このような時間シフトは，周期的シフト，循環シフトなどと呼ばれることもある．

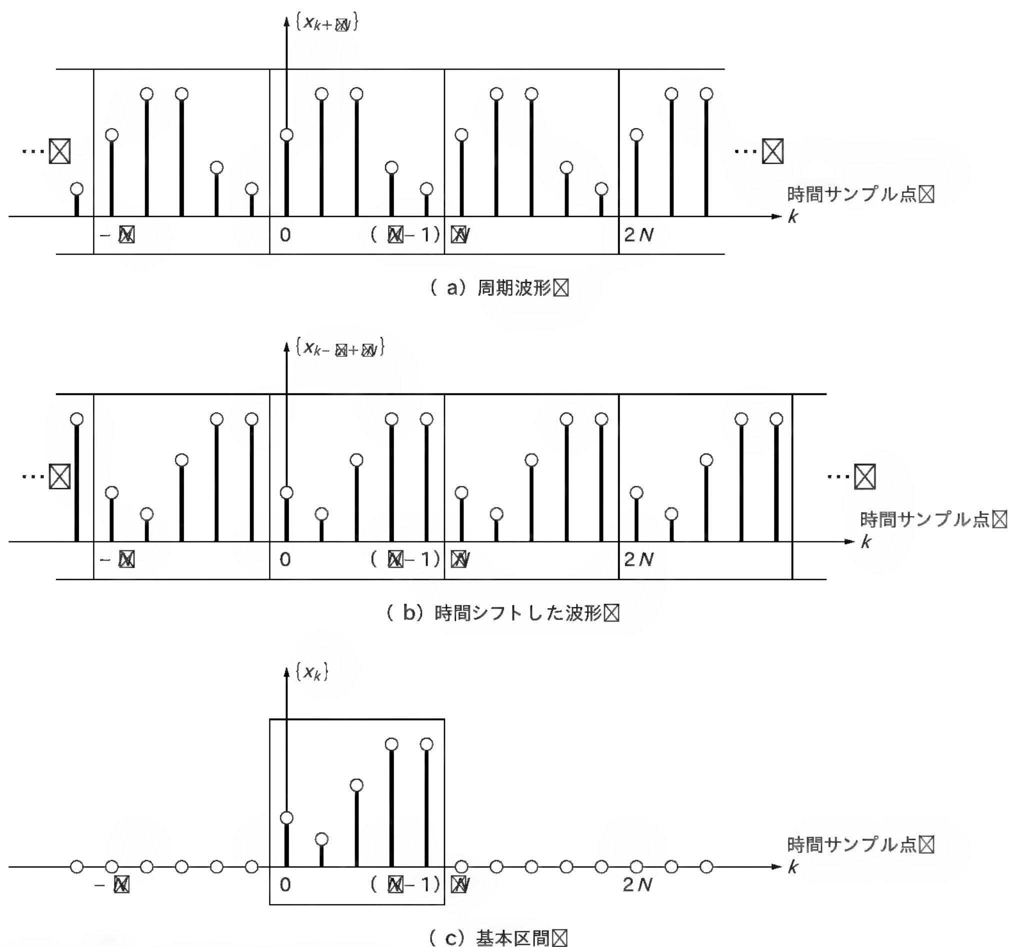


図 26.2 時間シフト($N=5$，$m=2$の場合)

よって，時間シフトした信号$\{x_{k-m}\}_{k=0}^{k=N-1}$のDFT値$\{X_\ell^{(-m)}\}_{\ell=0}^{\ell=N-1}$は，DFTの定義式より，$\ell=0, 1, 2, \cdots, (N-1)$に対して，

$$X_\ell^{(-m)} = \frac{1}{N}\sum_{k=0}^{N-1} x_{k-m}W_N^{k\ell} = \underbrace{\left[\frac{1}{N}\sum_{k=0}^{N-1} x_{k-m}W_N^{(k-m)\ell}\right]}_{X_\ell}\times W_N^{\ell m} \quad \cdots\cdots (18)$$

となり，

$$X_\ell^{(-m)} = X_\ell\times W_N^{\ell m} = X_\ell e^{-j\frac{2\pi}{N}\ell m} \quad \cdots\cdots (19)$$

が成立する．また，左へシフトしたときの信号$\{x_{k+m}\}_{k=0}^{k=N-1}$に対しては，

$$X_\ell^{(m)} = X_\ell\times W_N^{-\ell m} = X_\ell e^{j\frac{2\pi}{N}\ell m} \quad \cdots\cdots (20)$$

となる．

(iv) 時間反転

ディジタル信号$\{x_{k+rN}\}_{k=0}^{k=N-1}$の時間軸を反転させた信号，すなわち，

$$\{x_{-k+rN}\}_{k=0}^{k=N-1} \quad \cdots\cdots (21)$$

を考える[図26.3(a)，(b)]．このとき，添え字$(-k)$について式(17)と同様のmod演算を適用することにより，

$$x_{-k} = x_{N-k} \quad \cdots\cdots (22)$$

となり，DFT値$\{\tilde{X}_\ell\}_{\ell=0}^{\ell=N-1}$は，

$$\tilde{X}_\ell = \frac{1}{N}\sum_{k=0}^{N-1} x_{-k}W_N^{k\ell} = \frac{1}{N}\sum_{k=0}^{N-1} x_{N-k}W_N^{k\ell} \quad (N-k\text{を}n\text{と置く})$$
$$= \frac{1}{N}\sum_{n=0}^{N-1} x_n W^{(N-n)\ell} = \frac{1}{N}\sum_{n=0}^{N-1} x_n W_N^{-n\ell}$$
$$= \frac{1}{N}\sum_{n=0}^{N-1} x_n W_N^{n\times(-\ell)} = X_{-\ell} \quad \cdots\cdots (23)$$

と表される[図26.3(c)]．

例題1

いま，4サンプルのディジタル信号$\{x_k\}_{k=0}^{k=3}$のDFT値が，

$$X_0 = 2,\ X_1 = 1+j,\ X_2 = 3,\ X_3 = 1-j \quad \cdots\cdots (24)$$

であるとき，右へ1サンプルだけ時間シフトした波形のDFT値を求めよ．

解答1

式(19)において，$m=1$，$N=4$として計算すればよいので，

$$W_4 = e^{-j\frac{2\pi}{4}} = \cos\left(\frac{2\pi}{4}\right) - j\sin\left(\frac{2\pi}{4}\right) = -j$$



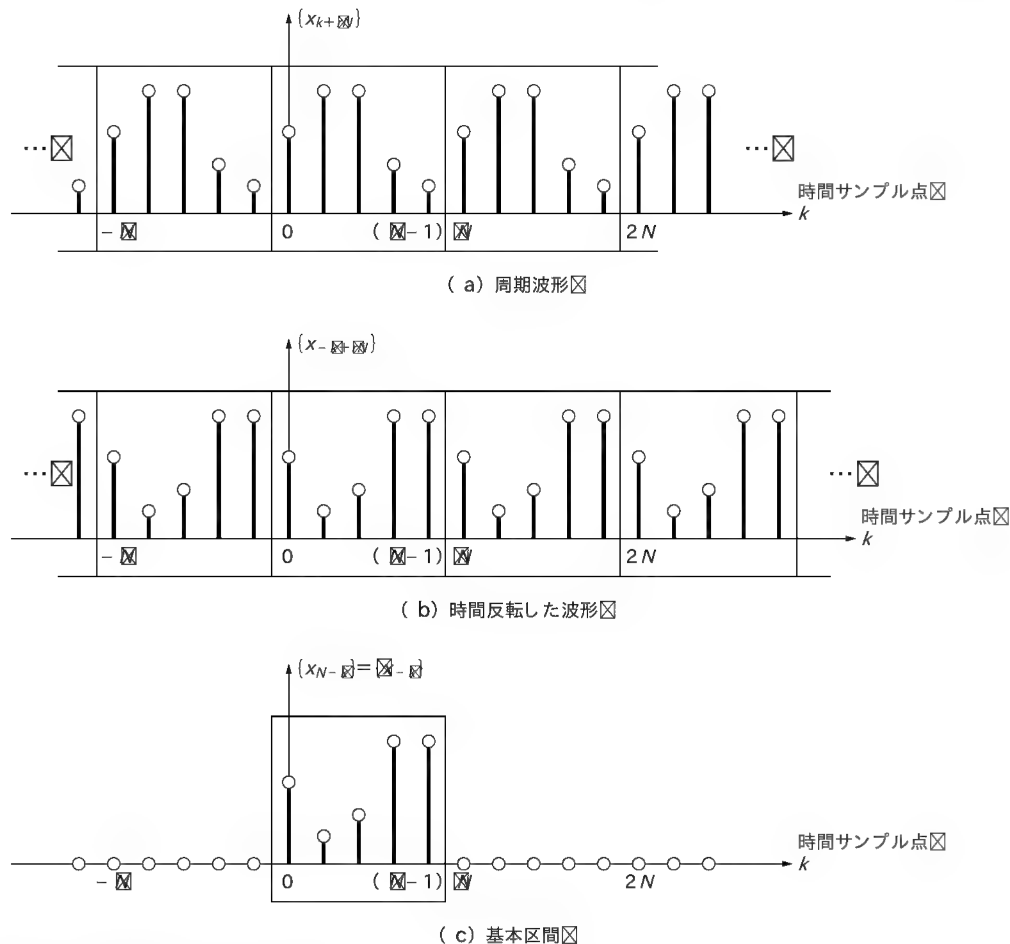


図26.3 時間反転(N＝5の場合)

を代入し，以下の結果が得られる．

$$
\begin{cases}
X_0^{(-1)} = X_0 \times W_4^{0} = X_0 = 2 \\
X_1^{(-1)} = X_1 \times W_4^{1\times1} = X_1 \times (-j) = 1-j \\
X_2^{(-1)} = X_2 \times W_4^{2\times1} = X_0 \times (-j)^2 = -3 \\
X_3^{(-1)} = X_3 \times W_4^{3\times1} = X_3 \times (-j)^3 = 1+j
\end{cases}
\quad \cdots\cdots\cdots\cdots\cdots (25)
$$

なお，時間シフトする前の信号は図26.4(a)であり，右へ1サンプルだけ時間シフトした波形は図26.4(b)となるので直接DFT値を計算して同様の結果［式(24)，式(25)］が得られることを検証しておいてほしい．

例題2

いま，ディジタル信号$\{x_k\}_{k=0}^{k=N-1}$が実数値波形であれば，そのDFT値$\{X_\ell\}_{\ell=0}^{\ell=N-1}$について，以下の関係が成立することを証明せよ．ただし，DFT値を実数部$\{R_\ell\}_{\ell=0}^{\ell=N-1}$と虚数部$\{I_\ell\}_{\ell=0}^{\ell=N-1}$に分けて，$X_\ell = R_\ell + jI_\ell$と表すものとする．

① 実数部は偶関数であること

② 虚数部は奇関数であること

解答2

基本的に，$\overline{X_\ell} = X_{-\ell}$であることを証明すればよい．式(11)よ

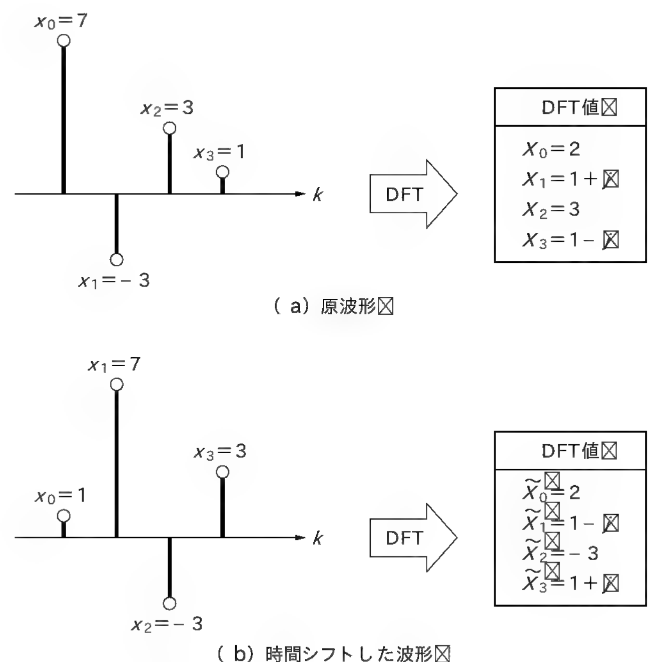


図26.4 例題1

り，両辺の複素共役を採って，

$$\overline{X_\ell}=\overline{\frac{1}{N}\sum_{k=0}^{N-1}x_k W_N^{k\ell}}$$
$$=\frac{1}{N}\sum_{k=0}^{N-1}x_k \overline{W_N^{k\ell}}$$
$$=\frac{1}{N}\sum_{k=0}^{N-1}x_k W_N^{-k\ell}$$
$$=\frac{1}{N}\sum_{k=0}^{N-1}x_k W_N^{k(-\ell)}=X_{-\ell} \quad\cdots\cdots(26)$$

となる関係が得られる．よって，式(26)より，

$$X_{-\ell}=\overline{R_\ell+jI_\ell}=R_\ell-jI_\ell \quad\cdots\cdots(27)$$

なので，$X_{-\ell}=R_{-\ell}+jI_{-\ell}$ より，

$R_{-\ell}=R_\ell$ （実数部は偶関数）$\cdots\cdots$(28)

$I_{-\ell}=-I_\ell$ （虚数部は奇関数）$\cdots\cdots$(29)

であることがわかる．

相関関数とDFT

さて，前回の「総まとめII(DFT，FFT編)」で解説済みの周期的畳み込み演算に類似した計算式で信号波形の性質を記述する関数の一つとして“相関関数”が知られている(詳細は，2002年1月号の第6回「DFTによる基本的な信号分析」を参照)．

相関関数は，ある信号波形を一定時間ずらしたときに，ずらす前の元の信号波形とどの程度の関連があるのかとか，類似しているのかを測る尺度であり，関連度あるいは類似度といえるパラメータである．

たとえば，2種類のディジタル信号$\{x_k\}_{k=0}^{k=N-1}$，$\{y_k\}_{k=0}^{k=N-1}$に対する関連度は，整数nに対して，

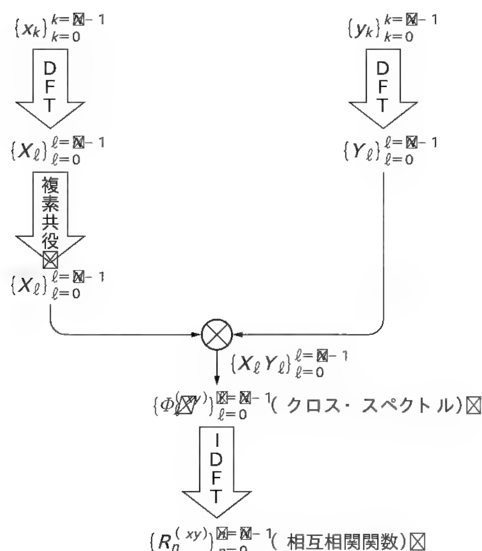


図26.5　DFTを用いた相互相関関数の算出手順

●相互相関関数

$$R_n^{(xy)}=\frac{1}{N}\sum_{k=0}^{N-1}x_k y_{k+n} \quad\cdots\cdots(30)$$

として定義される．ここで，y_{k+n}はサンプルに対する周期的シフトを行う．また，式(30)を二つの信号の1サンプルあたりの平均振幅値で正規化した値は，

$$\tilde{R}_n^{(xy)}=\frac{\frac{1}{N}\sum_{k=0}^{N-1}x_k y_{k+n}}{\sqrt{\frac{1}{N}\sum_{k=0}^{N-1}x_k^2}\sqrt{\frac{1}{N}\sum_{k=0}^{N-1}y_k^2}} \quad\cdots\cdots(31)$$

となり，相互相関係数とよばれる．

また，単一のディジタル信号$\{x_k\}_{k=0}^{k=N-1}$に対しての関連度は，整数nに対して，

●自己相関関数

$$R_n^{(xx)}=\frac{1}{N}\sum_{k=0}^{N-1}x_k x_{k+n} \quad\cdots\cdots(32)$$

が定義される．なお，式(32)を1サンプルあたりのエネルギで正規化した値$\tilde{R}_n^{(xx)}$は，

$$\tilde{R}_n^{(xx)}=\frac{\frac{1}{N}\sum_{k=0}^{N-1}x_k x_{k+n}}{\frac{1}{N}\sum_{k=0}^{N-1}x_k^2} \quad\cdots\cdots(33)$$

となり，自己相関係数とよばれる．

● DFTによるクロス・スペクトル(相互相関関数)の計算

まず，式(30)の相互相関関数$\{R_n^{(xy)}\}_{n=0}^{n=N-1}$のDFT値$\{\phi_\ell^{(xy)}\}_{\ell=0}^{\ell=N-1}$は“クロス・スペクトル”とも呼ばれ，

$$\phi_\ell^{(xy)}=\frac{1}{N}\sum_{n=0}^{N-1}R_n^{(xy)}W_N^{n\ell} \quad\cdots\cdots(34)$$

であり，式(30)を代入して以下のように式変形できる．

$$\phi_\ell^{(xy)}=\frac{1}{N}\sum_{n=0}^{N-1}\left[\frac{1}{N}\sum_{k=0}^{N-1}x_k y_{k+n}\right]W_N^{n\ell}$$
$$=\frac{1}{N}\sum_{n=0}^{N-1}\left[\frac{1}{N}\sum_{k=0}^{N-1}x_k y_{k+n}\right]W_N^{(n+k-k)\ell}$$
$$=\frac{1}{N}\sum_{k=0}^{N-1}x_k W_N^{-k\ell}\underbrace{\left[\frac{1}{N}\sum_{n=0}^{N-1}y_{k+n}W_N^{(k+n)\ell}\right]}_{Y_\ell}$$
$$=Y_\ell\underbrace{\left[\frac{1}{N}\sum_{k=0}^{N-1}x_k W_N^{-k\ell}\right]}_{\overline{X_\ell}}$$
$$=\overline{X_\ell}Y_\ell \quad\cdots\cdots(35)$$

よって，式(34)より，相互相関関数$\{R_n^{(xy)}\}_{\ell=0}^{\ell=N-1}$は，クロス・スペクトル$\{\phi_\ell^{(xy)}\}_{\ell=0}^{\ell=N-1}$のIDFT値として，

$$R_n^{(xy)}=\sum_{\ell=0}^{N-1}\phi_\ell^{(xy)}W_N^{-n\ell} \quad\cdots\cdots(36)$$

で与えられることになる(図26.5)．



● DFTによるパワー・スペクトル(自己相関関数)の計算

相互相関関数の説明の際に用いた信号y_{k+n}を，

$$y_{k+n} = x_{k+n} \quad \cdots\cdots (37)$$

と置けば，まったく同じ議論となるので，式(34)，式(35)，式(36)に基づき，

$$\phi_\ell^{(xx)} = \overline{X_\ell} X_\ell = |X_\ell|^2 \quad \cdots\cdots (38)$$

$$R_n^{(xx)} = \sum_{\ell=0}^{N-1} \phi_\ell^{(xx)} W_N^{-n\ell} \quad \cdots\cdots (39)$$

$$= \sum_{\ell=0}^{N-1} |X_\ell|^2 W_N^{-n\ell} \quad \cdots\cdots (40)$$

となる関係が導かれる(図26.6)．このとき，式(38)は信号$\{x_k\}_{k=0}^{k=N-1}$のDFT値の周波数ごとのパワー(平均電力)を表していることから，自己相関関数のDFT値$\{\phi_\ell^{(xx)}\}_{\ell=0}^{\ell=N-1}$は"パワー・スペクトル(またはピリオドグラム)"とも呼ばれる．とくに$n=0$の場合は，式(40)より，

$$R_0^{(xx)} = \sum_{\ell=0}^{N-1} |X_\ell|^2 \quad \cdots\cdots (41)$$

となり，さらには式(32)より，

$$R_0^{(xx)} = \frac{1}{N} \sum_{k=0}^{N-1} x_k^2 \quad \cdots\cdots (42)$$

であることから，1サンプルあたりの電力に一致することも理解できる．

例題3

図26.7のディジタル信号の自己相関関数$R_n^{(xx)}$，自己相関係数$\tilde{R}_n^{(xx)}$を，整数$n=(-10)\sim10$に対して計算し，変化のようすをグラフで示せ．

解答3

式(32)，式(33)に基づいて計算した結果を図26.8に示しておくので，各自で検証してもらいたい．図26.8より，自己相関関数は周期性を有することが確認される．

ここで，式(32)の自己相関関数$R_n^{(xx)}$には，簡単な計算により，

$$R_0^{(xx)} \geqq R_n^{(xx)} \quad \cdots\cdots (43)$$

$$R_n^{(xx)} = R_{-n}^{(xx)} \quad \cdots\cdots (44)$$

となる関係が成立することが導かれる．よって，式(43)，式(44)より，

「自己相関関数の最大値R_{max}が$R_0^{(xx)}$であること」

「自己相関関数が偶関数，すなわち$n=0$に対して線対称になること」

がわかる．また，式(33)，式(43)より，

「自己相関係数の最大値$\tilde{R}_{max}$は1に等しいこと」

になる．以上の三つの性質は，図26.8のグラフからも明らかである．

例題4

図26.9(a)，(b)のディジタル信号の自己相関関数$R_n^{(xx)}$を，整数$n=(-3)\sim3$に対して計算し，変化のようすをグラフで示せ．また，二つのディジタル信号のDFT値も計算せよ．

解答4

式(32)の計算結果として，自己相関関数$R_n^{(xx)}$を図26.10，DFT値〔式(1)〜式(4)を利用〕を図26.11に示しておくので，

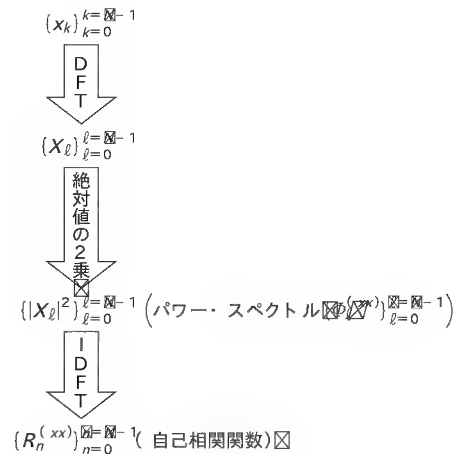


図26.6 自己相関関数とパワー・スペクトル

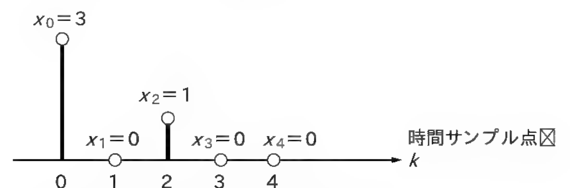


図26.7 例題3のディジタル信号$\{x_k\}_{k=0}^{k=4}$

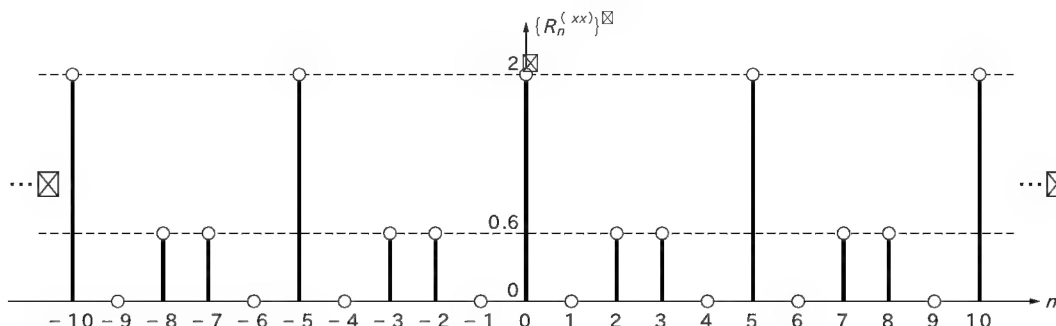


図26.8 例題3の自己相関関数

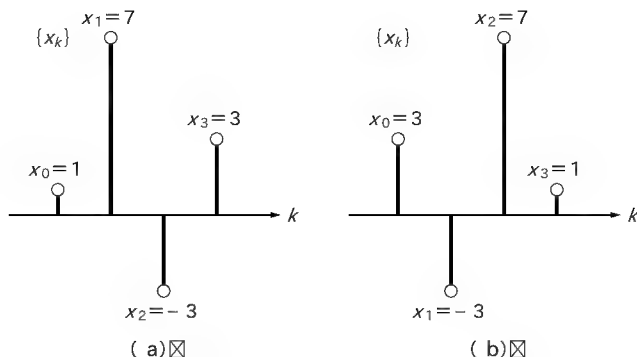


図 26.9　例題 4 のディジタル信号 $\{x_k\}_{k=0}^{k=3}$

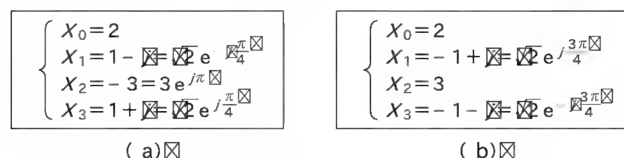
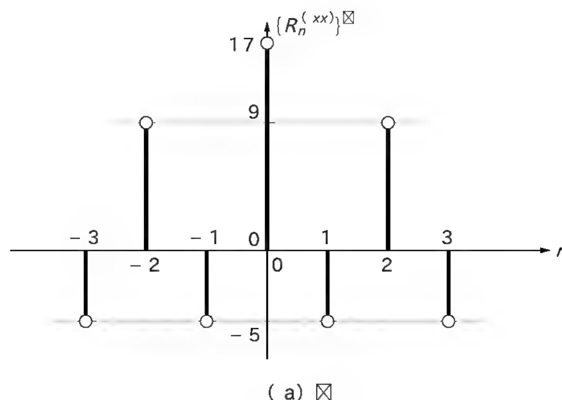


図 26.11　例題 4 の DFT 値



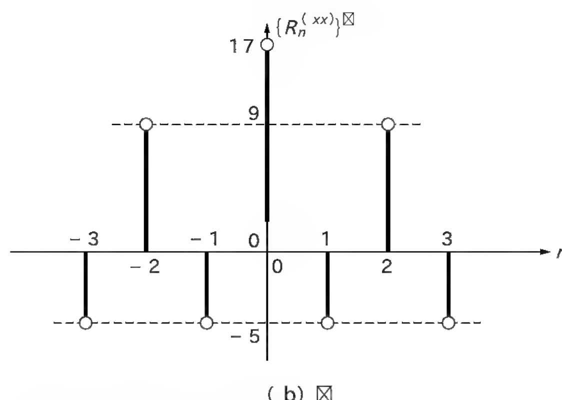


図 26.10　例題 4 の自己相関関数

各自で検証してもらいたい．

ところで，図 26.11 の二つの DFT 値を比較してみると，振幅は同じで位相だけが異なっていることに気づかれるであろう．そのため信号波形の形状は大きく異なってはいるが，図 26.10 からわかるように同一の自己相関関数を有しており，信号の位相に依存しないことがわかるのである．また，式(41)の関係が成立することも検証できる．

正規直交基底行列による画像表現

静止画像 S は通常 2 次元データとして，

$$S=\{s_{m,n}\}_{m,n=0}^{m,n=N-1} \quad \cdots\cdots (45)$$

で与えられる．いま，2 次元の正規直交基底行列 $\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=N-1}$ を，

$$\boldsymbol{\lambda}^{(k,\ell)}=\begin{bmatrix} \lambda_{0,0}^{(k,\ell)} & \lambda_{0,1}^{(k,\ell)} & \lambda_{0,2}^{(k,\ell)} & \cdots & \lambda_{0,(N-1)}^{(k,\ell)} \\ \lambda_{1,0}^{(k,\ell)} & \lambda_{1,1}^{(k,\ell)} & \lambda_{1,2}^{(k,\ell)} & \cdots & \lambda_{1,(N-1)}^{(k,\ell)} \\ \vdots & & \ddots & & \vdots \\ \lambda_{(N-1),0}^{(k,\ell)} & \lambda_{(N-1),1}^{(k,\ell)} & \lambda_{(N-1),2}^{(k,\ell)} & \cdots & \lambda_{(N-1),(N-1)}^{(k,\ell)} \end{bmatrix} \quad \cdots\cdots (46)$$

と表現し，さらに二つ行列の内積を，

$$\left\langle \boldsymbol{\lambda}^{(k,\ell)},\boldsymbol{\lambda}^{(m,n)}\right\rangle=\frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\lambda_{i,j}^{(k,\ell)}\lambda_{i,j}^{(m,n)} \quad \cdots\cdots (47)$$

と定義する．このとき，

$$\left\langle \boldsymbol{\lambda}^{(k,\ell)},\boldsymbol{\lambda}^{(m,n)}\right\rangle=\begin{cases}1\ ; & k=m,\ \ell=n \\ 0\ ; & k\neq m \text{ または } \ell\neq n\end{cases} \quad \cdots\cdots (48)$$

となる関係が成立すれば，

「$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=N-1}$ は“正規直交基底行列”をなす」

という．

ところで，1 次元の正規直交基底ベクトルの場合と同様に，正規直交基底行列 $\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=N-1}$ を用いることにより，式(45)の画像は，

$$\begin{aligned} S=\ & C_{0,0}\boldsymbol{\lambda}^{(0,0)}+C_{0,1}\boldsymbol{\lambda}^{(0,1)}+\cdots+C_{0,N-1}\boldsymbol{\lambda}^{(0,N-1)} \\ & +C_{1,0}\boldsymbol{\lambda}^{(1,0)}+C_{1,1}\boldsymbol{\lambda}^{(1,1)}+\cdots+C_{1,N-1}\boldsymbol{\lambda}^{(1,N-1)} \\ & \vdots \\ & +C_{N-1,0}\boldsymbol{\lambda}^{(N-1,0)}+C_{N-1,1}\boldsymbol{\lambda}^{(N-1,1)}+\cdots+C_{N-1,N-1}\boldsymbol{\lambda}^{(N-1,N-1)} \\ =\ & \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}C_{k,\ell}\boldsymbol{\lambda}^{(k,\ell)} \end{aligned} \quad \cdots\cdots (49)$$

と展開される．つまり，式(49)の各画素値 $\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$ は，

$$\begin{aligned} s_{m,n}=\ & C_{0,0}\lambda_{m,n}^{(0,0)}+C_{0,1}\lambda_{m,n}^{(0,1)}+\cdots+C_{0,N-1}\lambda_{m,n}^{(0,N-1)} \\ & +C_{1,0}\lambda_{m,n}^{(1,0)}+C_{1,1}\lambda_{m,n}^{(1,1)}+\cdots+C_{1,N-1}\lambda_{m,n}^{(1,N-1)} \\ & \vdots \\ & +C_{N-1,0}\lambda_{m,n}^{(N-1,0)}+C_{N-1,1}\lambda_{m,n}^{(N-1,1)}+\cdots+C_{N-1,N-1}\lambda_{m,n}^{(N-1,N-1)} \end{aligned} \quad \cdots\cdots (50)$$

と表される．

このとき，式(49)の展開係数 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$ は，

$$C_{k,\ell}=\left\langle S,\boldsymbol{\lambda}^{(k,\ell)}\right\rangle \quad \cdots\cdots (51)$$

のように，画像行列 S と正規直交基底行列 $\boldsymbol{\lambda}^{(k,\ell)}$ との内積として求められる．



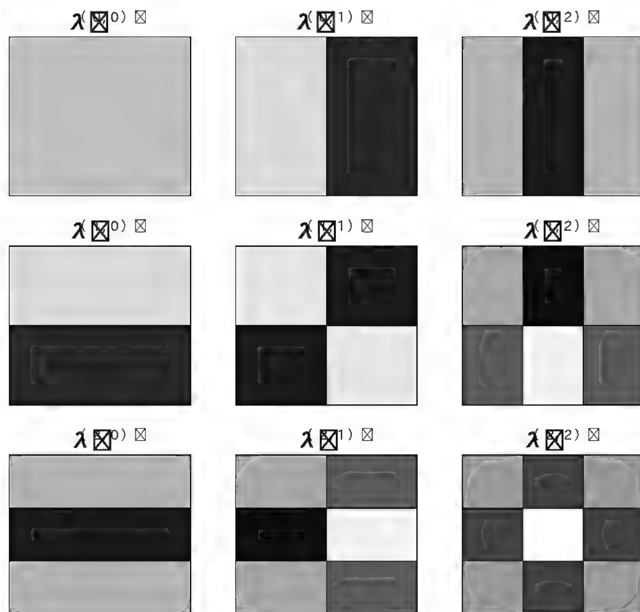


図 26.12　基底画像の例（3×3画素の場合）

2次元 DCT の基底画像

さて，式（46）～式（48）の条件を満たす $N\times N$ の正規直交基底行列 $\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=N-1}$ の代表例として，

$$\lambda_{m,n}{}^{(k,\ell)}=\gamma_k\gamma_\ell\cos\left[\frac{(2m+1)k}{2N}\pi\right]\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\cdots\cdots(52)$$

がある［m，n = 0，1，2，…，(N − 1)］．ただし，

$$\gamma_k,\ \gamma_\ell=\begin{cases}1\ ;\ k=0\ \text{または}\ \ell=0\\ \sqrt{2}\ ;\ k\neq 0\ \text{または}\ \ell\neq 0\end{cases}\cdots\cdots(53)$$

である．ここで，式（52），式（53）で与えられる正規直交基底行列は2次元 DCT の基底画像と称されるものであり，画像の分解・合成の基本になる．このとき，式（51）において，式（45）と式（52）の各行列の要素を用いて，

$$C_{k,\ell}=\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\gamma_k\gamma_\ell s_{m,n}\cos\left[\frac{(2m+1)k}{2N}\pi\right]\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\cdots\cdots(54)$$

と表され，2次元 DCT と呼ばれる．

それでは，N = 3を例に正規直交基底行列で表される 9（= 3×3）種類の基底画像 $\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=2}$ を示しておく（図 26.12）．同時に，式（51）で $S=\boldsymbol{\lambda}^{(k,\ell)}$ として，基底画像に対する2次元 DCT 値 $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=2}$，すなわち，

$$C_{k,\ell}=\left\langle\boldsymbol{\lambda}^{(k,\ell)},\boldsymbol{\lambda}^{(k,\ell)}\right\rangle\cdots\cdots(55)$$

も示す．

● 直流画像

$$①\ \boldsymbol{\lambda}^{(0,0)}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\Rightarrow\{C_{k,\ell}\}=\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix}$$

● 縦じま

$$②\ \boldsymbol{\lambda}^{(0,1)}=\begin{bmatrix}\frac{\sqrt6}{2}&0&-\frac{\sqrt6}{2}\\\frac{\sqrt6}{2}&0&-\frac{\sqrt6}{2}\\\frac{\sqrt6}{2}&0&-\frac{\sqrt6}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&1&0\\0&0&0\\0&0&0\end{bmatrix}$$

$$③\ \boldsymbol{\lambda}^{(0,2)}=\begin{bmatrix}\frac{\sqrt2}{2}&-\sqrt2&\frac{\sqrt2}{2}\\\frac{\sqrt2}{2}&-\sqrt2&\frac{\sqrt2}{2}\\\frac{\sqrt2}{2}&-\sqrt2&\frac{\sqrt2}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&1\\0&0&0\\0&0&0\end{bmatrix}$$

● 横じま

$$④\ \boldsymbol{\lambda}^{(1,0)}=\begin{bmatrix}\frac{\sqrt6}{2}&\frac{\sqrt6}{2}&\frac{\sqrt6}{2}\\0&0&0\\-\frac{\sqrt6}{2}&-\frac{\sqrt6}{2}&-\frac{\sqrt6}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\1&0&0\\0&0&0\end{bmatrix}$$

$$⑤\ \boldsymbol{\lambda}^{(2,0)}=\begin{bmatrix}\frac{\sqrt2}{2}&\frac{\sqrt2}{2}&\frac{\sqrt2}{2}\\-\sqrt2&-\sqrt2&-\sqrt2\\\frac{\sqrt2}{2}&\frac{\sqrt2}{2}&\frac{\sqrt2}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\0&0&0\\1&0&0\end{bmatrix}$$

● 格子じま

$$⑥\ \boldsymbol{\lambda}^{(1,1)}=\begin{bmatrix}\frac{3}{2}&0&-\frac{3}{2}\\0&0&0\\-\frac{3}{2}&0&\frac{3}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$$

$$⑦\ \boldsymbol{\lambda}^{(1,2)}=\begin{bmatrix}\frac{\sqrt3}{2}&-\sqrt3&\frac{\sqrt3}{2}\\0&0&0\\-\frac{\sqrt3}{2}&\sqrt3&-\frac{\sqrt3}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\0&0&1\\0&0&0\end{bmatrix}$$

$$⑧\ \boldsymbol{\lambda}^{(2,1)}=\begin{bmatrix}\frac{\sqrt3}{2}&0&-\frac{\sqrt3}{2}\\-\sqrt3&0&\sqrt3\\\frac{\sqrt3}{2}&0&-\frac{\sqrt3}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\0&0&0\\0&1&0\end{bmatrix}$$

$$⑨\ \boldsymbol{\lambda}^{(2,2)}=\begin{bmatrix}\frac{1}{2}&-1&\frac{1}{2}\\-1&2&-1\\\frac{1}{2}&-1&\frac{1}{2}\end{bmatrix}\Rightarrow\begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix}$$

以上の結果から，縦じま（②，③）と横じま（④，⑤）の各基底画像の要素ごとの積を計算してみると，格子じま（⑥～⑨）に等しくなることがわかる（各自で検証してもらいたい）．具体的には，

$$\begin{cases}\boldsymbol{\lambda}^{(1,1)}\ (⑥)：\text{縦じま}\ \boldsymbol{\lambda}^{(0,1)}\ (②)\ \text{と横じま}\ \boldsymbol{\lambda}^{(1,0)}\ (④)\\\boldsymbol{\lambda}^{(1,2)}\ (⑦)：\text{縦じま}\ \boldsymbol{\lambda}^{(0,2)}\ (③)\ \text{と横じま}\ \boldsymbol{\lambda}^{(1,0)}\ (④)\\\boldsymbol{\lambda}^{(2,1)}\ (⑧)：\text{縦じま}\ \boldsymbol{\lambda}^{(0,1)}\ (②)\ \text{と横じま}\ \boldsymbol{\lambda}^{(2,0)}\ (⑤)\\\boldsymbol{\lambda}^{(2,2)}\ (⑨)：\text{縦じま}\ \boldsymbol{\lambda}^{(0,2)}\ (③)\ \text{と横じま}\ \boldsymbol{\lambda}^{(2,0)}\ (⑤)\end{cases}$$

である．たとえば，$\boldsymbol{\lambda}^{(0,1)}$ と $\boldsymbol{\lambda}^{(1,0)}$ の要素ごとの積を計算すると，

$$\begin{bmatrix} \frac{\sqrt{6}}{2}\times\left(\frac{\sqrt{6}}{2}\right) & 0\times\left(\frac{\sqrt{6}}{2}\right) & -\frac{\sqrt{6}}{2}\times\left(\frac{\sqrt{6}}{2}\right) \\ \frac{\sqrt{6}}{2}\times(0) & 0\times(0) & -\frac{\sqrt{6}}{2}\times(0) \\ \frac{\sqrt{6}}{2}\times\left(-\frac{\sqrt{6}}{2}\right) & 0\times\left(-\frac{\sqrt{6}}{2}\right) & -\frac{\sqrt{6}}{2}\times\left(-\frac{\sqrt{6}}{2}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{2} & 0 & -\frac{3}{2} \\ 0 & 0 & 0 \\ -\frac{3}{2} & 0 & \frac{3}{2} \end{bmatrix} = \boldsymbol{\lambda}^{(1,1)}$$

となる(ほかの格子じまも同様).

また，基底画像$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=2}$とその2次元DCT $\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=2}$の値から，

$$\begin{cases} \text{添え字}k\text{は“横じまの零交差する数”} \\ \text{添え字}\ell\text{は“縦じまの零交差する数”} \end{cases}$$

を表すことも類推される(図26.13)．つまり，画像の周波数という概念が，基底画像や2次元DCT値を表す添え字(k, ℓ)に相当するのである．

例題5

いま，3×3画素の画像データ$S=\{s_{m,n}\}_{m,n=0}^{m,n=2}$を，

$$S=\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} \\ s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2,0} & s_{2,1} & s_{2,2} \end{bmatrix} = \begin{bmatrix} 253 & 208 & 199 \\ 220 & 196 & 172 \\ 205 & 184 & 163 \end{bmatrix} \quad\cdots\cdots(56)$$

とするとき，図26.12の正規直交基底画像$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=2}$を用いて表せ．

解答5

基底画像を用いて表すための展開係数は，画像$\{s_{m,n}\}_{m,n=0}^{m,n=2}$の2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=N-1}$に等しいことを利用する．式(54)より，

$$\begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} 200 & 8\sqrt{6} & 2\sqrt{2} \\ 6\sqrt{6} & 2 & 2\sqrt{3} \\ 2\sqrt{2} & 0 & 2 \end{bmatrix} \quad\cdots\cdots(57)$$

となり，式(49)に代入して，

$$\begin{bmatrix} 253 & 208 & 199 \\ 220 & 196 & 172 \\ 205 & 184 & 163 \end{bmatrix}$$

$$=200\times\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + 8\sqrt{6}\times\begin{bmatrix} \frac{\sqrt{6}}{2} & 0 & -\frac{\sqrt{6}}{2} \\ \frac{\sqrt{6}}{2} & 0 & -\frac{\sqrt{6}}{2} \\ \frac{\sqrt{6}}{2} & 0 & -\frac{\sqrt{6}}{2} \end{bmatrix}$$

$$+2\sqrt{2}\times\begin{bmatrix} \frac{\sqrt{2}}{2} & -\sqrt{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\sqrt{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\sqrt{2} & \frac{\sqrt{2}}{2} \end{bmatrix} + 6\sqrt{6}\times\begin{bmatrix} \frac{\sqrt{6}}{2} & \frac{\sqrt{6}}{2} & \frac{\sqrt{6}}{2} \\ 0 & 0 & 0 \\ -\frac{\sqrt{6}}{2} & -\frac{\sqrt{6}}{2} & -\frac{\sqrt{6}}{2} \end{bmatrix}$$

$$+\ 2\ \times\begin{bmatrix} \frac{3}{2} & 0 & -\frac{3}{2} \\ 0 & 0 & 0 \\ -\frac{3}{2} & 0 & \frac{3}{2} \end{bmatrix} + 2\sqrt{3}\times\begin{bmatrix} \frac{\sqrt{3}}{2} & -\sqrt{3} & \frac{\sqrt{3}}{2} \\ 0 & 0 & 0 \\ -\frac{\sqrt{3}}{2} & \sqrt{3} & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

$$+2\sqrt{2}\times\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} + 0\times\begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & -\frac{\sqrt{3}}{2} \\ -\sqrt{3} & 0 & \sqrt{3} \\ \frac{\sqrt{3}}{2} & 0 & -\frac{\sqrt{3}}{2} \end{bmatrix}$$

$$+\ 2\ \times\begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -1 & 2 & -1 \\ \frac{1}{2} & -1 & \frac{1}{2} \end{bmatrix} \quad\cdots\cdots(58)$$

となり，図26.12の正規直交基底画像$\{\boldsymbol{\lambda}^{(k,\ell)}\}_{k,\ell=0}^{k,\ell=2}$による展開式が得られる．

例題6

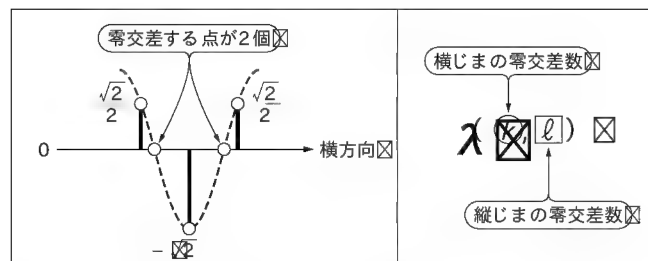
例題5 の画像データ$\{s_{m,n}\}_{m,n=0}^{m,n=2}$について，1サンプルあたりの平均エネルギ，すなわち，

$$P=\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left(s_{m,n}\right)^2 \quad\cdots\cdots(59)$$

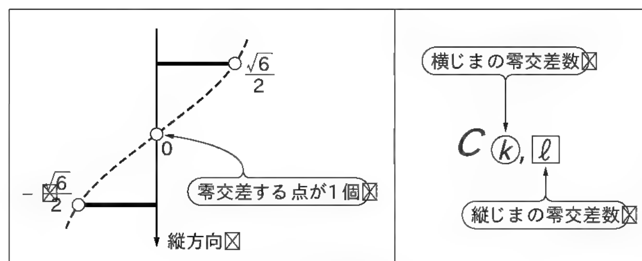
を求めよ．さらに，2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=2}$より，

$$P=\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}\left(C_{k,\ell}\right)^2 \quad\cdots\cdots(60)$$

を計算し，式(59)の計算値に一致することを示せ(パーシバルの定理)．



(a) $\boldsymbol{\lambda}^{(2,2)}$の場合



(b) $C_{1,0}$($\boldsymbol{\lambda}^{(1,0)}$に相当)の場合

図26.13　基底画像，2次元DCT値の零変差数



解答6

● 式(59)による計算値

$$P=\frac{1}{3^2}\left(253^2+208^2+199^2+220^2+196^2+172^2+205^2+184^2+163^2\right)=40636$$

● 式(60)による計算値

$$P=200^2+\left(8\sqrt{6}\right)^2+\left(2\sqrt{2}\right)^2+\left(6\sqrt{6}\right)^2+2^2+\left(2\sqrt{3}\right)^2+\left(2\sqrt{2}\right)^2+0^2+2^2=40636$$

ここで，画像データ，あるいは2次元DCT値から求めた1サンプルあたりの平均エネルギPが一致することを簡単に説明する．式(49)，式(50)，式(59)より，

$$P=\langle S,S\rangle=\frac{1}{N^2}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left(s_{m,n}\right)^2=\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}\sum_{k'=0}^{N-1}\sum_{\ell'=0}^{N-1}C_{k,\ell}C_{k',\ell'}\left\langle \boldsymbol{\lambda}^{(k,\ell)},\boldsymbol{\lambda}^{(k',\ell')}\right\rangle$$

と表され，さらに式(48)の正規直交関係を適用すれば，容易に式(60)に一致することが導かれる．

2次元DCT，IDCTによる画像圧縮・再合成

まず，2次元逆ディジタル・コサイン変換(これ以後，2次元IDCTと記す)の一般式を示す．

$$s_{m,n}=\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1}\gamma_k\gamma_\ell C_{k,\ell}\cos\left[\frac{(2m+1)k}{2N}\pi\right]\cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \qquad (61)$$

なお，IDFTと同様に，式(61)の2次元DCT計算式は，式(54)の2次元DCTを$N\times N$個の未知変数$\{s_{m,n}\}_{m,n=0}^{m,n=N-1}$に対する連立方程式とみなしたときの解に相当している．

それでは，**例題5**の3×3画素の画像データを例に，JPEGやMPEGで利用される画像データの圧縮・再合成の基本的な考え方を説明してみよう(図26.14)．

最初に，画像$\{s_{m,n}\}_{m,n=0}^{m,n=2}$の2次元DCT値$\{C_{k,\ell}\}_{k,\ell=0}^{k,\ell=2}$，すなわち正規直交基底行列の展開係数を計算する〔式(57)の再掲〕．

$$\begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix}=\begin{bmatrix} 200 & 8\sqrt{6} & 2\sqrt{2} \\ 6\sqrt{6} & 2 & 2\sqrt{3} \\ 2\sqrt{2} & 0 & 2 \end{bmatrix} \qquad (62)$$

となる．

また，画像のブロック化したデータは同じような値をとる(直流成分が多く含まれる)のが一般的である．そこで，式(62)において，たとえば，しきい値を5として，2次元DCT値のうち5未満のもの，すなわち，

$C_{0,2}$，$C_{1,1}$，$C_{1,2}$，$C_{2,0}$，$C_{2,1}$，$C_{2,2}$

をすべて零(0)にするのである．この強制的に零(0)にする処理が"画像データの圧縮"に相当し，JPEGやMPEGでの基本的な処理につながる考え方である．すると，圧縮された2次元DCT値$\{\tilde{C}_{k,\ell}\}_{k,\ell=0}^{k,\ell=2}$は，

$$\begin{bmatrix} \tilde{C}_{0,0} & \tilde{C}_{0,1} & \tilde{C}_{0,2} \\ \tilde{C}_{1,0} & \tilde{C}_{1,1} & \tilde{C}_{1,2} \\ \tilde{C}_{2,0} & \tilde{C}_{2,1} & \tilde{C}_{2,2} \end{bmatrix}=\begin{bmatrix} 200 & 8\sqrt{6} & 0 \\ 6\sqrt{6} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad (63)$$

となる．

次に，式(63)の2次元DCT値から，もとの画像を再合成してみることにしよう．式(61)の2次元IDCTの計算式を適用すればよいので，再合成した画像データを$\tilde{S}=\{\tilde{s}_{m,n}\}_{m,n=0}^{m,n=N-1}$と表せば，簡単な計算により，

$$\begin{bmatrix} \tilde{s}_{0,0} & \tilde{s}_{0,1} & \tilde{s}_{0,2} \\ \tilde{s}_{1,0} & \tilde{s}_{1,1} & \tilde{s}_{1,2} \\ \tilde{s}_{2,0} & \tilde{s}_{2,1} & \tilde{s}_{2,2} \end{bmatrix}=\begin{bmatrix} 242 & 218 & 194 \\ 224 & 200 & 176 \\ 206 & 182 & 158 \end{bmatrix} \qquad (64)$$

となる．ここで，2次元DCT値のうち小さいものを強制的に零(0)にする処理にともない，再合成した画像$\tilde{S}$はもとの画像Sに完全には一致しないことに注意してほしい．なお，再合成時の誤差ΔSは，式(56)と式(64)の差として与えられるわけで，

$$\Delta S=\tilde{S}-S=\begin{bmatrix} -11 & 10 & -5 \\ 4 & 4 & 4 \\ 1 & -2 & -5 \end{bmatrix} \qquad (65)$$

となる．

このように，2次元DCTを用いて周波数成分ごとに分解して，小さい値を強制的に零にすることにより，再合成時の画像の劣化を極力少なくする形で，"画像データの圧縮"を可能にするのである．

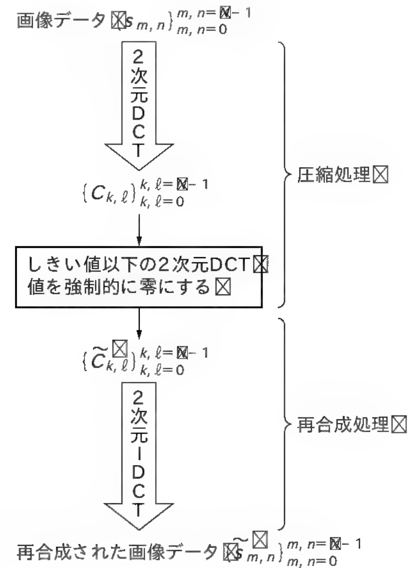
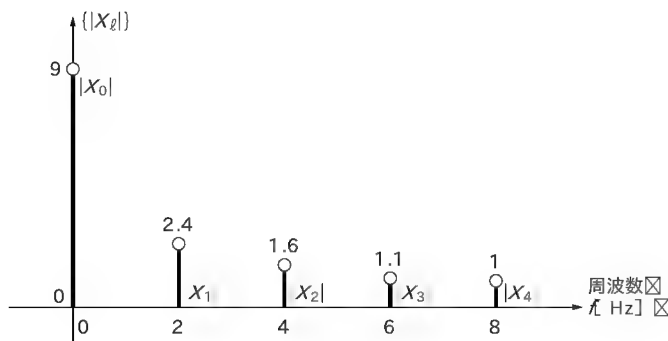
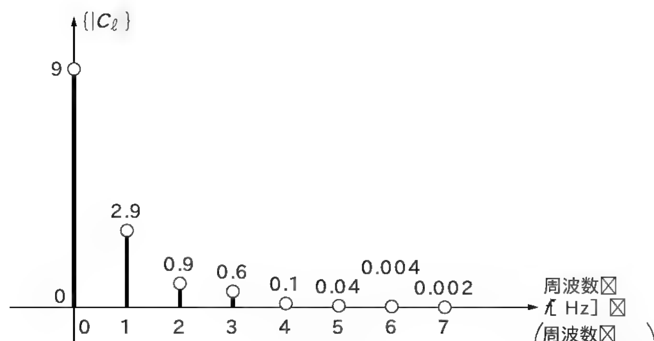


図26.14 画像データの圧縮・再合成の処理手順



（a）DFT値



（b）DCT値

図26.15　信号スペクトルの変化のようす

DFTとDCTの適用ポイント

最後に，本連載「信号数学」の締めくくりとして，DFTとDCTの着目すべき適用ポイントをまとめておこう．

DFTとDCTはいずれも，基本的に周波数成分を分析するための手法であるわけだが，いくつか知っておきたい違いがあるので紹介しておきたい．

とりあえず，サンプル数N[個]のDFTとDCTの計算式を示すことから始めよう．

$$(\mathrm{DFT})\ X_\ell=\frac{1}{N}\sum_{k=0}^{N-1}x_k\left(e^{-j\frac{2\pi}{N}}\right)^{k\ell}$$

$$=\frac{1}{N}\sum_{k=0}^{N-1}x_k\left\{\cos\left(\frac{2\pi k\ell}{N}\right)-j\sin\left(\frac{2\pi k\ell}{N}\right)\right\}\cdots\cdots(66)$$

$$(\mathrm{DCT})\ G_\ell=\frac{1}{N}\sum_{k=0}^{N-1}\gamma_\ell x_k\cos\left[\frac{(2k+1)\ell}{2N}\pi\right]\cdots\cdots\cdots\cdots(67)$$

- DFTは複素数演算，DCTは実数演算

式(66)，式(67)から明らかであり，計算量の観点からは，DCTが有利である．

- 信号の周波数分解能を高くするにはDCT

サンプリング周波数をf_s[Hz]とするとき，

$$\text{DFTの周波数分解能}=\frac{1}{N}\times f_s\,[\mathrm{Hz}]\cdots\cdots\cdots\cdots(68)$$

$$\text{DCTの周波数分解能}=\frac{1}{2N}\times f_s\,[\mathrm{Hz}]\cdots\cdots\cdots\cdots(69)$$

なので，DCTはDFTの2倍の周波数分解能を有している．

- 信号の位相情報を知るにはDFT

位相情報を知るには，信号変換値が実数部と虚数部をもつことが必要であり，実数値のDCT値から読み取ることはできない（例題4参照）．

- 信号のスペクトルを集中させるにはDCT

いま，$N=8$サンプルのディジタル信号（サンプリング間隔$T=0.125$[秒]，すなわちサンプリング周波数$f_s=1/T=8$[Hz]とする）を，

$$\{x_k\}_{k=0}^{k=7}=\{0,\ 4,\ 8,\ 10,\ 11,\ 12,\ 13,\ 14\}\cdots\cdots\cdots\cdots(70)$$

として，DFTの絶対値$\{|X_\ell|\}_{\ell=0}^{\ell=7}$とDCT $\{G_\ell\}_{\ell=0}^{\ell=7}$をそれぞれ式(66)，式(67)を適用し，計算してみる（図26.15）．

図26.15より，DCTではスペクトルが低周波側に集中していることがわかる．これに対して，DFTはDCTに比べ高周波の成分も多いことが確認できる．そのため，画像のように変化の少ない性質を有する信号に対しては，直流成分が非常に大きくなることからDCTが有利となる．

＊　　　＊

今回をもって「信号数学」に関する解説を終えることになるわけだが，DFTとDCTはあらゆる信号の周波数分析ツールとして信号処理全般の中核に位置するものなので，読者のみなさんにはしっかりと内容を理解し，身につけておくことを強くお勧めしたい．

今後，頭のリフレッシュを兼ねてしばらく充電する期間をいただいた後，新たなテーマに挑戦したいとの想いをめぐらせている．現在のところ，次の連載では「やり直しのためのデータ処理数学（仮題）」として，本連載「信号数学」の内容を受ける形で，ウェーブレット変換，画像データ圧縮（JPEG，MPEG），符号化（turbo符号，算術符号）を採り上げてみたいと考えている．

さらには，通信における変調，復調，伝送などの信号処理で必要となる数学的知識を解説したいとも思う．もちろん，「信号数学」と同様にわかりやすい解説を心がけ，読者のみなさんのお役に立てればと考えている．どうぞお楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科

組み込みプログラミング・ノウハウ入門(第17回)

制約が厳しい条件でのスケジューリング

――ハード・リアルタイムはやっぱりCyclic Executive

藤倉 俊幸

1 はじめに

「処理自身は単純だが，時間制約が厳しく，システム・リソースも開発リソースも少ない」――そんな場合にどうやって組み込みソフトを作れば良いのか，それが今回のテーマである．

ここでいう「少ない」とは，以下のものである．

- 時間制約が厳しいというのは，処理のデッドラインが決められている，いわゆる「ハード・リアルタイム・システム」で，しかもデッドラインが数ms程度でかなり短いという意味である
- システム・リソースが少ないというのは，CPUパワーとメモリが最小限で，RTOSを使うことをあきらめるしかないということである
- 開発リソースが少ないというのは，開発期間が短く予算もないので開発メンバの教育をしている余裕もないということである
- 処理が単純とは，ほとんど決まりきった処理の周期的な繰り返しであり，枯れたアプリケーションなので処理自身に関する仕様変更もほとんどないという意味である

● 時間駆動方式

このような状況でよく使われるテクニックに時間駆動方式がある．

図1は，スペースシャトルのナビゲーション・ソフトウェアの構造[1]を示したものである．制御の要であるExecutiveと，処理の要であるFunctional Processing Moduleに，周期的に処理が起動される時間駆動型モジュールが使用されている．また，ナビゲーション・ソフト以外のサブシステムも時間駆動方式が使われており，結論として，ハード・リアルタイム部分は基本的に時間駆動型で作られているといえる．

ちなみにナビゲーション・ソフトのサイズは全部で160Kバイト程度である．今では信じられないくらい，1980年代のハードウェア環境は厳しかった．その後，RTOSがあたりまえの時代になって時間駆動方式は忘れ去られたような感がある．

ところが最近，たとえばCPUにH8を使ってRTOSなしでシステムを作りたいというような案件がいくつか出てきた．このような案件は機器制御が中心で，時間駆動方式が向いている．最近あちこちで開催されているロボット競技の環境もほとんどはH8レベルである．

UMLフォーラムのロボコン[2]を見学したときに，図2のような走り方をするロボットがいくつかあった．走っているうちに徐々に蛇行幅が大きくなってしまう走り方である．走行速度とセンサの振り方などのタイミングが合っていないのではないか？と思われる．ロボコンでは設計資料が展示されているものの，タイミング的なことはほとんど表現されていないので，どのよ

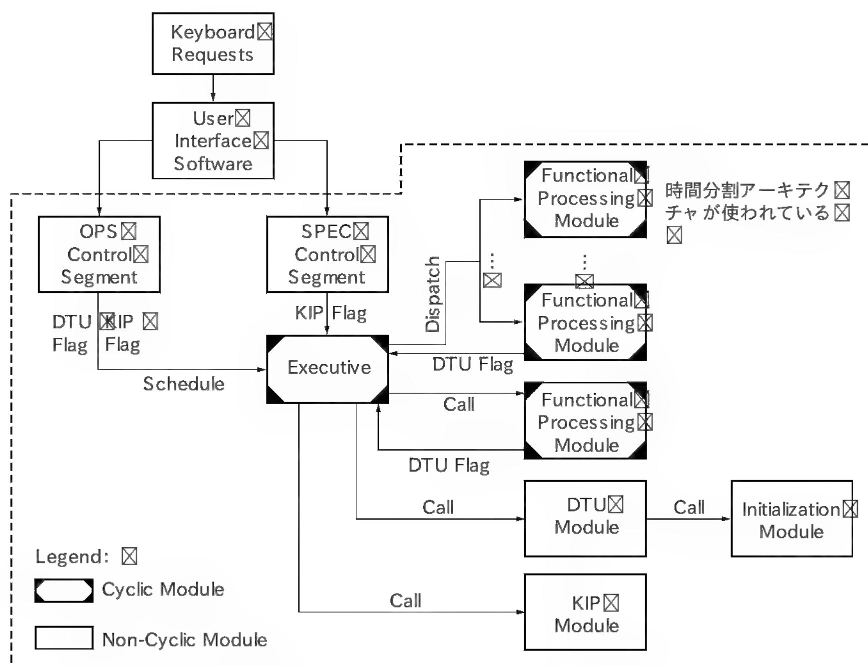


図1 スペースシャトルのナビゲーション・ソフトウェアの構造

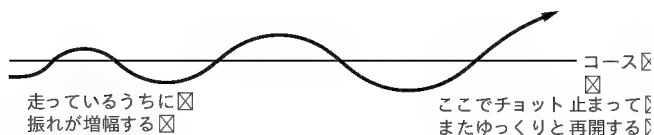


図2　ロボットの走行動作

うな制御を行っているのか，肝心のところがわからないのが残念だ．

ロボコンよりもハイレベルな競技にロボワン(ROBO-ONE)[3]がある．Web[4]で動画を見ることができる．こちらはかなり詳細な技術資料がロボワン公式サイトや参加者のWebサイトで公開されている．ソフトウェアのタイミング設計の資料は少ないので詳細はわからないが，時間駆動方式を採用しているロボットが多いようである．ただし，ソフトウェア・アーキテクチャとしては，先にタイム・スロットを決めてしまい，そこに制御プログラムを分割して割り当てていくことが多いようである．サーボ・モータの制御を最優先させるためにモータの仕様からタイム・スロット幅が決められてしまい，モータ制御の空き時間にそのほかの処理ソフトを走らせる構造になっている．ここで，後ほど解説するCyclic Executiveというアーキテクチャを使用すればもう少しソフトウェア設計の自由度が増えるのではないかと思う．

さらにハイレベルな自律分散型ロボットによるサッカー競技のロボカップ(RoboCup)[5]では，参加は大学の研究室が中心になる．技術資料は，各参加研究室のWebサイトである程度公開されている．動的にタイム・スロットを割り当てたりする高度な時間駆動方式のものもあるが，人工知能的な制御が中心で特に動的構造を意識していないロボットもあるようだ．ハードウェアもリッチになってSH-2クラス[6]を使用している．それにともなってRTOSを使用するようになっている．その結果，イベント・ドリブン型通信スレッドと時間駆動型のモータ制御スレッドの混在アーキテクチャ[7]が増えてくる．

ロボット以外でも，時間分割やタイム・スロットなどのキーワードを使ってWebで検索すると，ファイル・アクセスや通信方式，スケジューリングなどいろいろな分野における，「少ないリソースをやりくりしながら並行に処理を進める方法」がヒットする．携帯電話や自動車関係などの組み込みの中でも最近話題になる分野が多い．

時間駆動方式というとRTOS以前の古い技術のように感じるが，依然として有用な技術である．イベント駆動型に比べると，時間駆動型は不定期に発生する外部イベントに対してはポーリングする形になるので応答時間が長くなったり，ジッタが発生したり，CPU時間をむだに使うなど効率が悪い．しかし，システム全体を同期して動かすことができる場合には，非同期通信よりも同期通信のほうが通信速度を出せるように，またはDRAMよりSDRAMが高速なようにメリットもある．

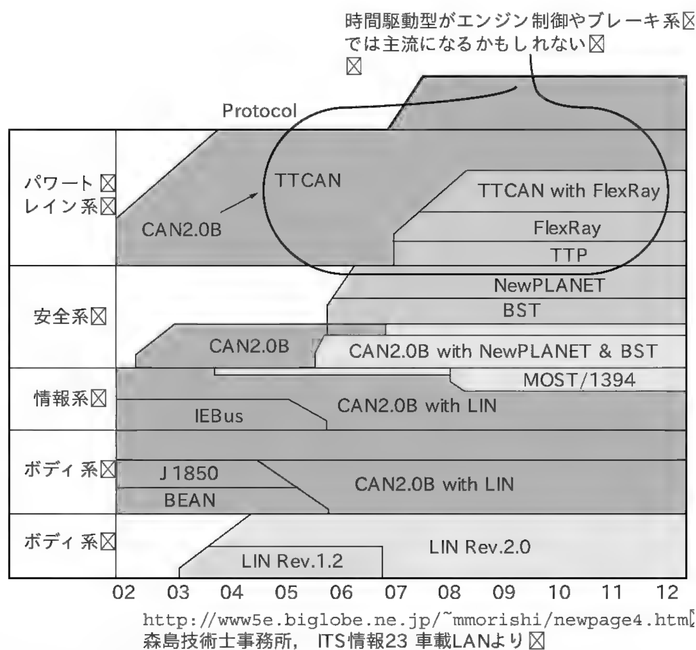


図3　将来の車載LAN

● 時間駆動とイベント駆動

自動車で利用される各種の機器を接続するための車載LAN用のプロトコルが各種提案されている(図3)．車載LANとしてはCAN(Controller Area Network)[8]が有名だが，CANの次のプロトコルと見られるものの中には時間駆動型を採用したプロトコルがいくつかある．たとえば，TTCAN(Time Triggered CAN)[9]，TTP(Time Triggered Protocol)[10]，FlexRay[11]などがある．時間駆動型の分散環境では，各ノードの処理もネットワーク・トラフィックもあらかじめ確保されたタイム・スロットに割り付けることでリアルタイム性を保証することができる．もっとも今回は，分散環境ではなく，基本となる一つのノードの中での静的なスケジューリングについて扱うこととする．Cyclic Executive[12]は，インターバル・タイマのみで構築できる，静的スケジューリングを利用するハード・リアルタイム用のアーキテクチャである．

2 RTOSを使わない動的アーキテクチャ

ソフトウェア・アーキテクチャということばもかなり一般的に使われるようになった．ここで扱うのは，その中で「スレッドの制御の方法」に関するアーキテクチャである．このアーキテクチャを特に動的アーキテクチャと呼ぶことにする．RTOSのAPIの使い方より，もう少し抽象度の高いところの話になる．

ソフトウェア・アーキテクチャについてはこの連載の3回目で取り上げた．そのときの後半で説明したものが動的アーキテクチャである(図4)．記事の中では汎用アーキテクチャと専用



図4　アーキテクチャの分類

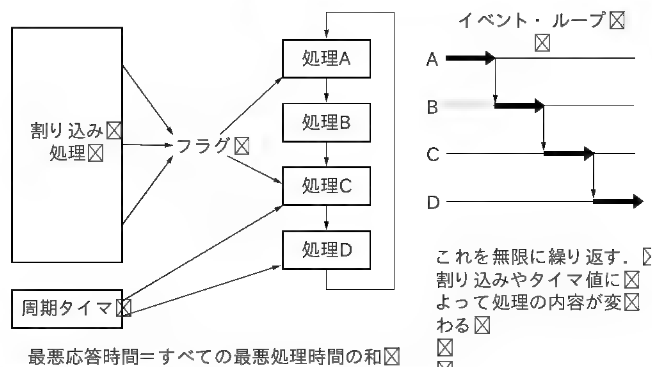


図5　イベント・ループ型（その1）

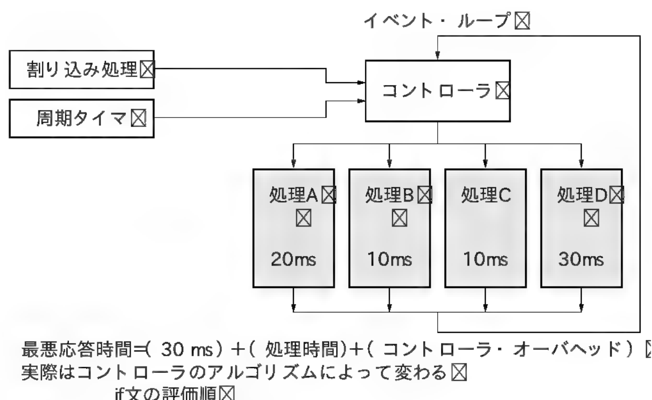


図6　イベント・ループ型（その2）

アーキテクチャで分けて解説していた．たとえば，パイプライン型アーキテクチャについては前半と後半で説明したが，前半のパイプライン型はデータあるいはオブジェクトが流れていくパイプライン型で，後半のパイプライン型はスレッド制御が流れていくパイプライン型である．名前は同じでも別物である．今風に言えば，そもそもアスペクトが異なっている．後半で説明したパイプライン型が動的アーキテクチャに属している．このとき，Cyclic Executive型も簡単に説明した．

● イベント・ループ型

RTOSを使わない場合によく使う動的構造はイベント・ループ型である．いくつかの割り込みハンドラとメイン関数があって，メイン関数は無限ループをしながら割り込みが入るのを待ち，割り込みが入ったことを知らせるフラグが立つと対応する処理ルーチンを呼び出すというものだ．

割り込みを使わずにメイン関数のみで構成し，各処理ルーチンがハードウェア・レジスタからデータを読み出す場合もある．この場合は，並行性で問題が生じることはないので，排他制御の必要もない．一方，割り込みを使う場合は，割り込みハンドラと割り込み処理ルーチン間で並行性が生じるので，割り込みを禁止して排他制御を行うことになる．

図5の方法では，割り込み検出フラグの評価を各割り込み処理ルーチンに任せている．割り込みハンドラと割り込み処理ルーチンをモジュールとして扱いがちだが，最悪応答時間は各処理時間の合計になってしまう．

図6の方法には，割り込み検出フラグの評価をまとめて行うことで応答時間を短くするとともに，割り込み処理に関してプライオリティを導入することができるという利点がある．プライオリティは，コントローラ部分の`if`文の評価順や`switch-case`文の`case`の並び方で実装する．RTOSを使う場合は各スレッドにプライオリティを与える．このプライオリティはスレッドの属性データとして与えるので，実行しながら変更することができる．しかし，イベント・ループ型の場合，プライオリティはハード・コードしてしまうので，コンパイル時までに決定してしまう必要がある．このようなスケジューリングは静的なスケジューリングとかオフライン・スケジューリングなどと呼ばれている．しかし，イベント・ループ型は確率的要素があり，スケジュールを確定することが難しい場合がある．

イベント・ループ型は，（インターバル・タイマ以外の）外部イベントでドライブする場合がほとんどなので，イベント駆動型のアーキテクチャに属する．したがって，システムのふるまいは外部事象に依存してしまう．一般に静的スケジューリングを利用するとシステムのふるまいを予測しやすいのだが，負荷状況やわずかなタイミングのズレでリアルタイム性を保証できないのがイベント駆動型の弱点だ．

3 Cyclic Executive型

Cyclic Executive型はインターバル・タイマのみで構成する．基本的に外部事象に依存しないので，ハード・リアルタイム・システム向きである．これは航空機の規格であるDO-178B(13)でも認められているアーキテクチャである．

欠点としては，スケジュールがハード・コードしてあるため，保守がたいへんになることである．特にメイン・ループにアイドル・タイムが少ない場合が危ない．これは，イベント・ループ型でも同様で，RTOSを導入してスレッドという概念に動的構造をまとめなければ避けられない問題である．たとえば，処理が追加になった場合やアルゴリズムが変更になって処理時間が増えた場合，アイドル・タイムがなくなってしまうと変更になった処理だけでなく，ほかの処理についても実行時間を調整

してシステム全体の静的スケジューリングをやり直さなければならない場合がある．つまり，ソフトウェアの論理的な構造と時間的な構造が分離されていないので，一部分の変更でも処理時間が大きく変わる場合は，変更がシステム全体へと連鎖的に及んでしまうのである．RTOSを導入しないと時間的な構造と論理的な構造を分離することは難しい．ただし，この欠点は，ツールの使用や，より上位の設計情報を再利用する開発方法などを導入することである程度カバーすることができる．

もう一つの欠点は，ハード・デッドラインを持った処理とソフト・デッドラインを持った処理が混在した場合の扱いである．高負荷時には，ソフト・デッドラインをミスしてもハード・デッドラインは守らなければならないが，Cyclic Executiveでは，このようなふるまいを行わせることが難しい．すべてをハード・デッドラインとして扱ってしまうことになる．また，外的要因による高負荷状態を検出することもできない．こちらの欠点は，かなり本質的なもので，事実上Cyclic Executiveはハード・デッドライン専用と考えたほうが良い．

もし，ソフト・デッドラインと混在させる場合には，RTOSの導入などを検討する必要がある．その場合，Cyclic Executiveはシングル・スレッドとして実装できるので，高プライオリティの一つのタスクとしてRTOS下で実行させるとか，カーネルよりもプライオリティの高い別のプログラムとして実行させるなどでRTOSとの混在が可能である．混在させてソフト・デッドラインの部分をRTOSに分担させることになる．

● 前提条件

Cyclic Executiveを利用できるのは，基本的に周期的な処理に対してである．また，処理のデッドラインは起動周期よりも短い場合である．起動周期よりもデッドラインが長いとデータのバッファリングが必要となり，ふるまいの解析が複雑になるということがその理由である（めんどうだから嫌だと言っているようだが，本当にたいへんなのである）．データのバッファリングが必要ないような環境に適用できる．一般に，Cyclic ExecutiveはRAMをあまり使えない環境で利用するので，バッファを使えないということは大きな障害にならない場合が多い．

一つの処理について，cを最悪実行時間，dをデッドライン，pを起動周期とした場合に，

$$c \leqq d \leqq p \quad (1)$$

が成立する必要がある．このような処理は複数あっても良い．Cyclic Executiveは，このような複数の処理を一つのCPUで順番に実行するためのアーキテクチャである．処理の実行順は，すべてのデッドラインを守れることを確認してあらかじめ確定させて，プログラムにハード・コードしてしまう．

● メジャー・サイクルとマイナ・サイクル（サンプル1）

Cyclic Executiveにはメジャー・サイクルとマイナ・サイクルの二つの周期で構成する．

マイナ・サイクルが，タイム・スロット幅を決める．たとえば，図7のような周期処理が四つあってそれぞれの起動周期が25ms，50ms，200ms，1sの場合，Cyclic Executiveの制御部分に25msごとに起動をかける．これがマイナ・サイクルになる．制御部分は，これら四つの処理を25msごとに図8のように実行していく．

Task1は起動周期が25msでマイナ・サイクルと同一なので，マイナ・サイクルごとに実行されるが，そのほかのTask2～Task4はデッドラインをオーバしないように適当に組み合わされて実行される．必要な組み合わせは，図8に示した4種類である．これら4種類の実行パターン（フレームと呼ぶ）の実行時間が25ms以下であればデッドラインを保証できる．25ms以内に実行が終了した場合はアイドル状態になるかバックグラウンド処理を実行しながら，次のタイム・スロット開始を待つ．25ms以上実行時間がかかってしまった場合は，フレーム・オーバランと呼ぶ．このときは，オーバランする処理を分割してスケジュールし直すことになる．分割された処理をsliceとかstrip，chunk，schedulink blockと呼ぶ．

メジャー・サイクルは，タイム・スロットの繰り返しが一巡

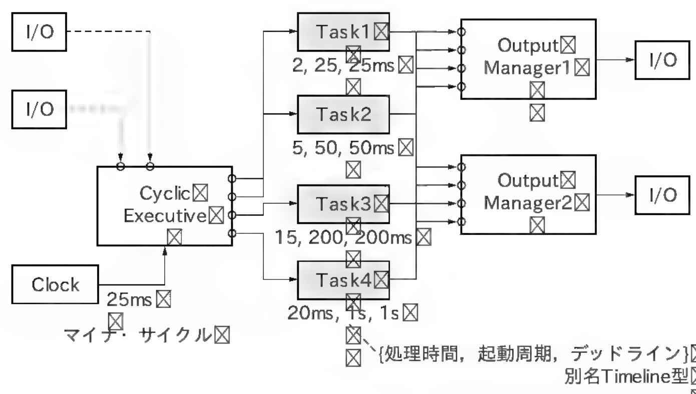


図7　Cyclic Executive

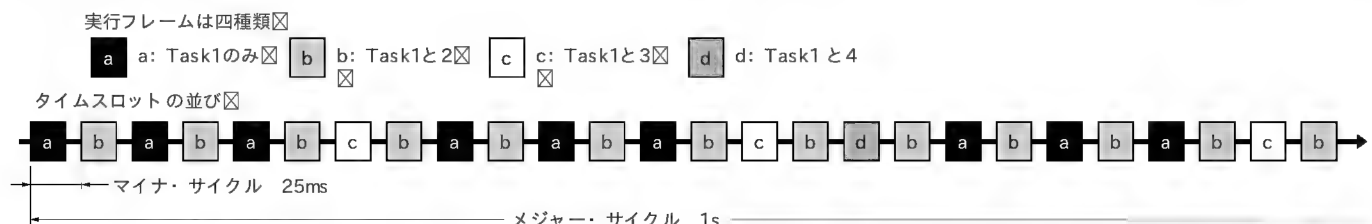


図8　Cyclic Executiveのスケジューリング

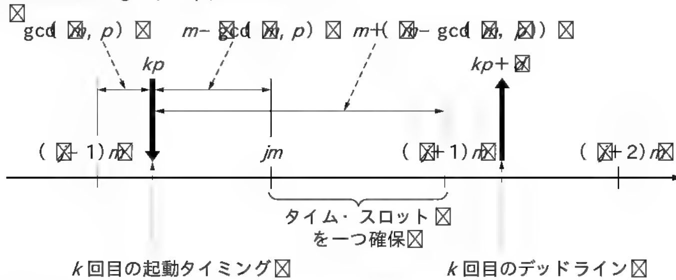


図9　マイナ・サイクル条件

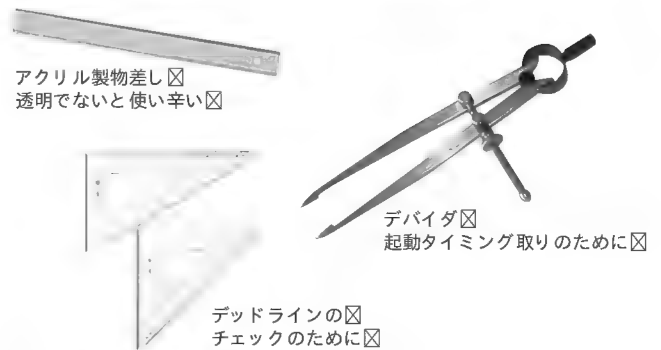


写真1　あると便利な道具

する周期である．メジャー・サイクルの長さは，各処理の周期の最小公倍数になる．図8の例では，25，50，200，1000の最小公倍数なので1sがメジャー・サイクルになる．マイナ・サイクルが25msなので，1000/25＝40サイクルで繰り返しパターンが一巡する．25msごとにスケジュール通りにパターンを進めるように制御部をコーディングする．通常は，処理関数へのポインタの配列を用意してタイマ割り込みのたびに一つずつ進めるパターンが取られる．

● マイナ・サイクルの決め方

一般に可能なマイナ・サイクルは複数ある．Cyclic Executiveで扱いたい処理がn個あって，それぞれの処理時間，起動周期，デッドラインを$\{(c_1, p_1, d_1), (c_2, p_2, d_2), \cdots, (c_n, p_n, d_n)\}$としたとき，マイナ・サイクル$m$が満足すべき条件は以下のようになる．

1) 最小のデッドライン以下にする

$$m \leqq d_i \quad (i = 1, \cdots, n) \quad \cdots\cdots (2)$$

2) 最大の実行時間以上にする

$$m \geqq c_i \quad (i = 1, \cdots, n) \quad \cdots\cdots (3)$$

3) メジャー・サイクルの約数にする

4) 各起動周期とデッドラインに対して次式が成立する(図9)

$$m + (m - \gcd(m, p_i)) \leqq d_i \quad (i = 1, \cdots, n) \quad \cdots\cdots (4)$$

最後の条件は，最悪のタイミングでタイム・スロットが始まった場合でもデッドラインまでに一つの実行フレームを確保するための条件である．ここで，$\gcd(m, p)$はmとpの最大公約数を表す．

● マイナ・サイクルの計算(サンプル2)

処理をA，B，Cの三種類とし，処理時間，起動周期，デッドラインをそれぞれ，A＝(1，14，14)，B＝(2，20，20)，C＝(3，22，22)とする．式(2)により，$m \leqq 14$，式(3)により，$m \geqq 3$である．この場合のメジャー・サイクルは，14，20，22の最小公倍数なので1540である．この約数は，1，2，4，5，7，10，11，14，20，22，…だが上記の条件を満足するのは，4，5，7，10，11，14となる．最後に式(4)を各pとdのペアに対して適用すると4，5，7が残る．この三つがマイナ・サイクル候補になる．短いほうが応答性は良くなりジッタも減るが，処理時間が収まりきれずに処理を分割しなければならなくなる確率が高くなるので注意が必要だ．

● スケジューリング

メジャー・サイクルとマイナ・サイクルが決まったら，タイム・スロットに処理を割り付けていく．注意しなければならないのは，上記の条件を満たしていても必ずスケジュールできるとは限らないことである．処理を分割しなければならない場合もある．実際に割り付けを行わないとわからないのである．それは，式(4)がタイム・スロット一つ分しか確保しないことが原因である．CPU使用率が高くなるケースではタイム・スロットが一つでは不足する場合がある．

基本的に処理の割り付けは，組み合わせ問題である．いわゆるNP-hard問題になるので，最適解をツールなどを使って簡単に求めることはあきらめたほうが良い．そこで，最適ではないかもしれないが，実行可能解をヒューリステックに求めることになる．この場合は，箱詰め(Bin-Packing)問題として知られているものと同じなので，この問題の近似解法を使用する場合が多い．市販のタイミング解析ツール(14)などは，独自のくふうによるアルゴリズムを実装している．

そのようなツールがない場合には，紙と鉛筆でスケジューリングを行う．紙と鉛筆以外に，物差しとかデバイダなどの製図用品が役に立つ(写真1)．ときどき，これらのアナログ機器(？)を使うとなぜか頭がリラックスする．マニュアルでスケジューリングする場合でも，起動周期の短いものから割り当てるとか，デッドラインの一番近いものから割り当てるなど，プリエンプティブなスケジューリング法と同じことを行えば何とかなる場合が多い．ただし，メジャー・サイクル分を割り当てなければならないのでチョット辛い作業である．たとえば，サンプル2の場合にはメジャー・サイクルが1540msなので，マイナ・サイクルに4msを選択した場合には384スロットを割り付けなければならない．制御部分のコーディング量も増えるという理由

で，マイナ・サイクルにはなるべく長いものを採用したくなる．

スケジュールが行き詰った場合は，割り当てられなかった処理を分割してスケジュールしていけば良い．CPU使用率が1％以下であれば必ず割り当ては終了する．スケジュールができたら，実際に分割可能かどうかを確認して制御部分にハード・コードする．処理を分割しなければならないところがCyclic Executiveの最大の欠点である．一般的には，起動周期の長いものほど処理時間も長いので，レート・モノトニック法をシミュレートして処理を割り付けていくと分割する確率が高くなる．分割する際には，共有リソースを使用していない箇所で分割しなければならないので，スケジュールした処理時間のところで分割できるとは限らない．

● 処理の分割（サンプル3）

処理をA，B，C，Dの四種類とし，処理時間，起動周期，デッドラインをそれぞれ，A＝(1，10，10)，B＝(3，10，10)，C＝(2，20，20)，D＝(8，20，20)とする．メジャー・サイクルは，10と20の最小公倍数で20になる．マイナ・サイクルmは，式(2)により，$m \leqq 10$．式(3)により，$m \geqq 8$．メジャー・サイクルの約数は，1，2，4，5，10，20で上記の条件を満足するのは，$m = 10$となる．式(4)も満足するので，マイナ・サイクルは10とする．

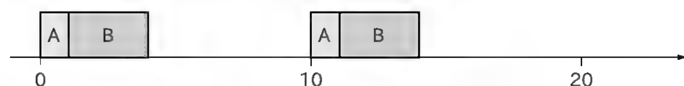
実際に割り当てを行うと，図11に示したようにDがデッドライン・オーバするので，DをD1とD2に分割する．分割条件は，D2の部分の処理時間が6ms以下であり，かつD1の処理時間が4ms以下になる．この条件の範囲で排他的な実行部分を避けて分割しなければならない．排他制御以外にもループ部分があったりするので，処理時間で分割するというのは，ソースコードを見てもなかなかわからないめんどうな作業である．

分割した処理D1とD2は，AとBのように独立な処理ではない．D1をD2の前に実行しなければならない順序制約が発生する．この順序制約の実装をCyclic Executiveの制御部に持ち込むと良くない．このような制約は，処理Dの中にカプセル化するようにする．具体的には，Cyclic Executiveから呼び出す関数は一つにして，その関数の中でD1とD2を順序制約に従って呼び分けるようにするのが良い．ついでにCのほうも2回に一度だけ実行するように内部で制御する．このようにすると，図10の(3)の二つの実行フレームは同一となってしまう．したがって，実装はなんと10msごとにA，B，C，Dを呼び出すだけの図5と同じになってしまう．順序制約のカプセル化には，ステート・マシンを利用しておけば変更の自由度が増す．

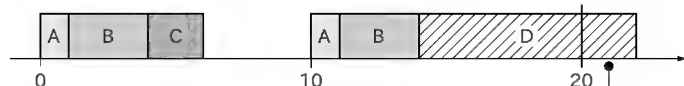
図11にイベント・ループ型(その2)で実行した場合を示す．イベント・ループ型その2では，起動タイミングのばらつきが発生しやすい．図11の場合では，処理Aは正確に10msごとに起動される．処理Bも位相はずれるが，ほぼ10msごとに起動される．変動要因は処理Aの処理時間の変動のみになる．一方，図11では，処理Aの起動がすでにかなり変動してしまう．

さらに，イベント・ループではほとんど同時に起動がかかるような場合の微妙なタイミングのズレで実行順が変動するので，つねにデッドラインを守れるとは限らない．

1）まず周期10のAとBを各スロットに割り当てる



2）次にCを第一スロットに割り当てる，Dを第二スロットに割り当てようとするとデッドライン・オーバする



3）そこでDを分割して割り当てる

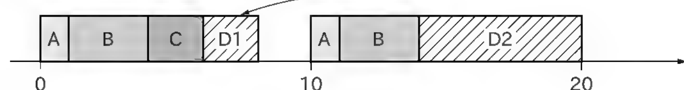
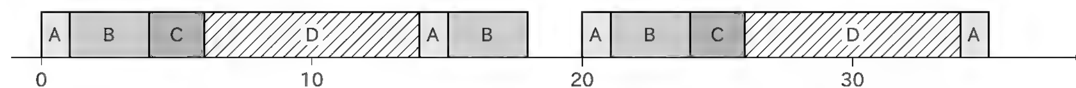


図10 サンプル3のスケジューリング（その1）

4 モード変更とオーバラン

組み込み機器には動作モードというものがある．例として，図12にスペースシャトルの動作モードを示す．発射前20分ま

A, B, C, Dの順にプライオリティ付けされている場合．デッドライン・オーバしないが
A, Bのジッタが大きくなる



ただしわずかなタイミングのズレで，AまたはBがデッドライン・オーバする可能性がある．非決定論的な挙動を示すようになる

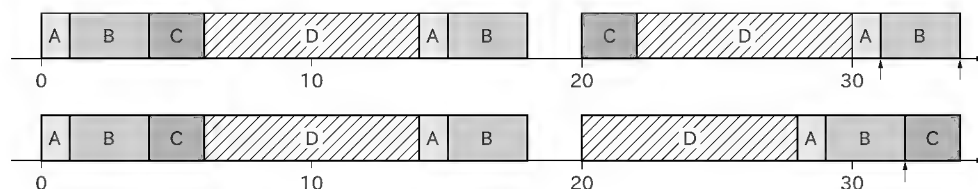


図11
サンプル3のイベント・ループ型（その2）

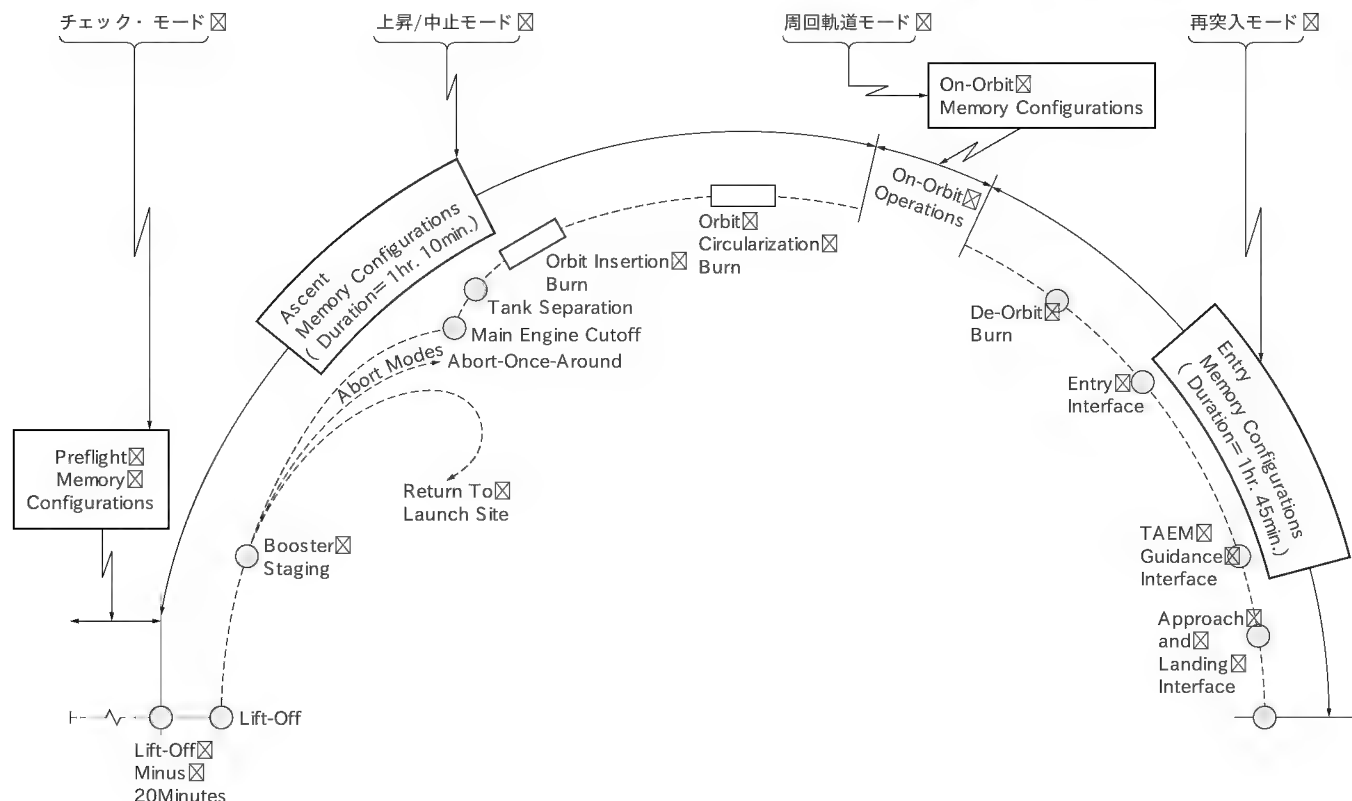


図12　スペースシャトルの動作モード

では点検用のプログラムが走っている．その後，周回軌道に乗るまで，周回軌道上，着陸の順で，スケジュールされる処理プログラム群が変更される．これを動作モード変更と呼ぶ．動作モードはこのような離散的なものだけとは限らない．エンジン制御のように，エンジンの回転数が1000rpmから7000rpmに変わる場合など，連続的に起動周期が変わらなければならない．また，連続的に起動周期を変えつつ，回転数に応じて排気ガスの組成が変動するので環境用の処理アルゴリズムが変更になるなどの離散的なモード変更も行わなければならなかったりする．このため，イベント駆動と時間駆動を使い分ける必要がある．

とりあえず離散的にモードが変更されるときには，Cyclic Executiveの制御部分が切り替わることになる．このときの実行中のフレームの扱いには，

1) ただちに中止して，切り替える
2) 実行中のマイナ・サイクルの最後まで実行してから切り替える
3) メジャー・サイクルの最後まで実行してから切り替える

などの選択肢がある．また，アイドル時に実行するバックグラウンド処理がある場合には，これも切り替える．

実行フレームの実行が，マイナ・サイクル内に終了しなかった場合の実行中のフレームの扱いには，モード変更の場合のほかに，

1) ただちに中止して，次のフレームを実行する
2) ただちに中断して，残りをバックグラウンド処理に回す
3) そのまま実行して，次のフレームの開始を遅らせる

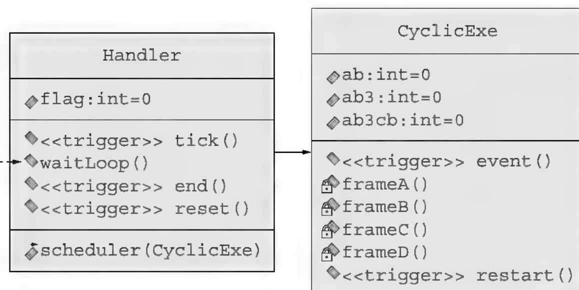
などの選択肢がある．2)の中断/再開を組み込むためには，ディスパッチャが必要になる．この場合，割り込み状態と通常状態，それからバックグラウンド状態でソフトウェアが動作することになり，簡易OSを組み込むのと同じことになる．また，残りの部分の実行の再スケジューリングとデッドライン保証などの問題があるのであまり現実的でない場合が多い．

Cyclic Executiveの制御部分は，関数ポインタの配列を用意して，タイマ割り込みが入るたびに関数を実行しながら関数ポインタを進めていけば良いので非常に簡単である．しかし，オーバランやモード変更の際のリカバ処理が実行中のフレームに依存している場合には，単純な配列構造では間に合わない場合がある．このような場合は，制御部分にも階層化ステート・マシンを利用したほうが良い．特に，システムが状態を持っていて，ある状態のときオーバランが発生したらホールドするが，別の状態でオーバランが発生した場合はリスタートするような場合はステート・マシンで実装すると良い．

たとえば，図7と図8であげたサンプル1のメジャー・サイクルをすべて書くと以下のような実行フレームの列になる．

a b a b a b c b a b a b a b a c b d a b a b a b c b a b a b a b c b a b

このCyclic Executiveの制御部分のステート・マシンによる




```
while( this->flag < 1) {
 while( this->flag == 0);
 event( this->scheduler);
 end( this );
}
```

図13　サンプル1の実装

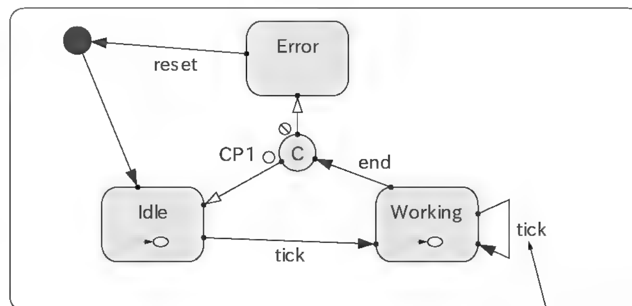


図15　Handlerのステート・マシン

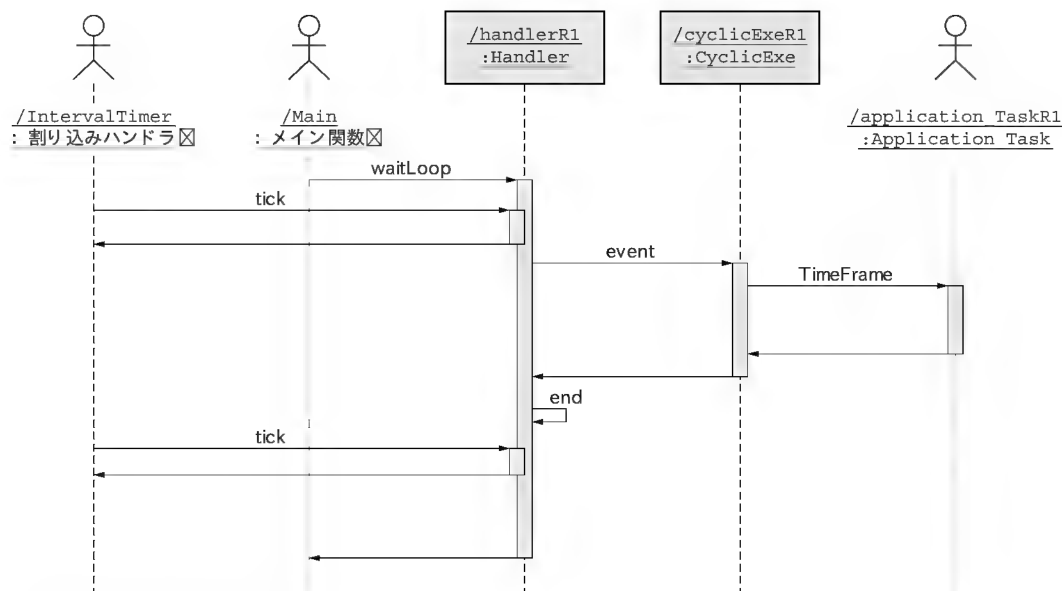


図14　割り込み状態との切り分け

実装を考えてみよう．

タイマ割り込みを利用するので，割り込み状態と通常状態を切り分ける必要がある．そこで，Handlerというクラスを使ってこの切り分けを行う．Handlerクラスのtick()関数は割り込みハンドラから呼び出される．一方，waitLoop()関数は通常状態から呼び出されて無限ループしながら割り込みフラグが立つたびにCyclicExeクラスのevent()を呼び出す．この関数はCyclicExeクラスのステート・マシンのトリガ関数になっている．図13にwaitLoop()のコードを示す．C言語で実装してあるので自分を指すthisポインタを使っている．これは，サンプル作りに利用したツールの制約である．tick()は，Handlerクラスが持っているステート・マシンのトリガ関数になっている(図15)．

図14に動作を示した．CyclicExeから呼び出しているApplication Taskが実際の周期処理を行う関数であり，既存のイベント・ループ型で利用していたものでも良い．図14で周期処理関数の実行が長くなってCyclicExeクラスのevent()から制御がwaitLoop()に戻ってend()を実行する前に，tick()が呼び出されるとオーバランが発生したことになる．それは，図15のWorking状態のtick自己遷移で処理される．Tickは割り込み状態で実行されるのでフラグを立てるだけで，実際のオーバラン処理はEnd遷移の中で行う．End遷移はwaitLoop()から呼び出される通常状態で実行される関数である．ここでオーバラン・フラグをチェックする．

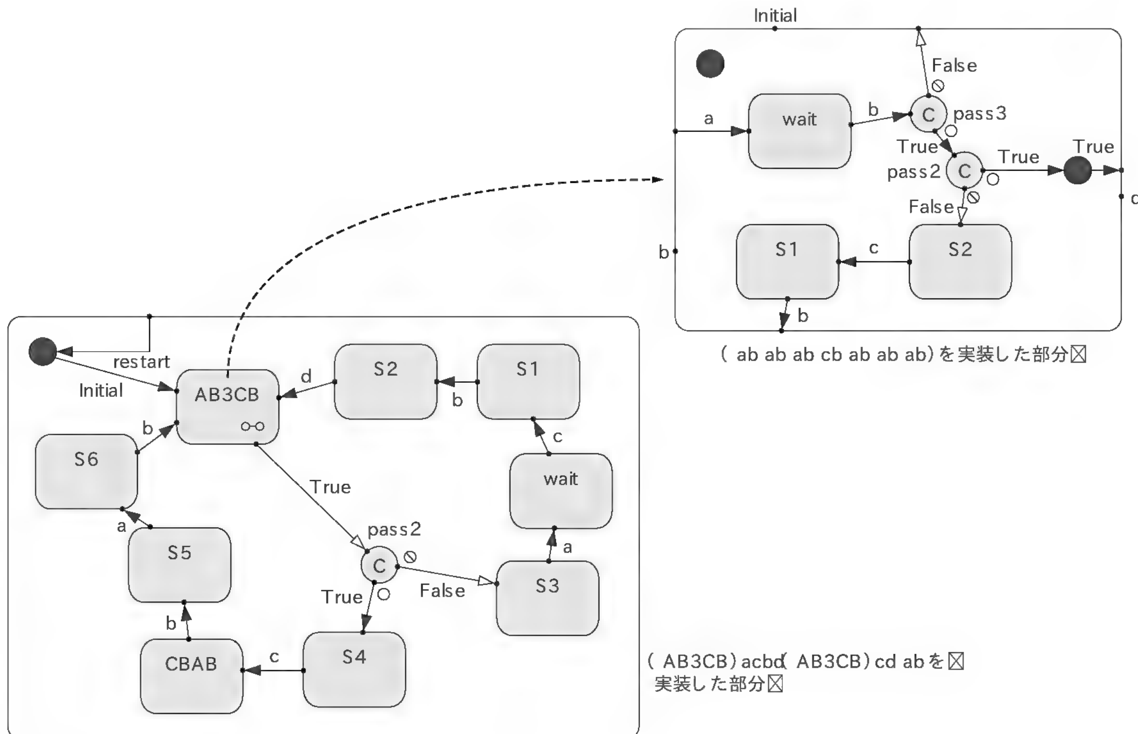


図16　CyclicExeのステート・マシン

静的に決定したスケジュールのハード・コードは，上記の実行フレーム列を，

(ab ab ab cb ab ab ab) acbd(ab ab ab cb ab ab ab) cb ab

のようにグループ分けして，

(AB3CB) acbd(AB3CB) cb ab

AB3CB =(ab ab ab cb ab ab ab)

の二つの階層化ステート・マシンで実装した(図16)．必ずしも，ここでステート・マシンを使う必要はないが，特定のタイムスロット実行中にオーバランなどのエラーが起こったときに特定の動作を行わせたい場合には，ステート・マシンで実装しておくことで対応が簡単になる．あるいは，デバッグ用またはプロトタイプとしてスケジュール実装部分をステート・マシンで作っておいて最終的に関数ポインタ型にする方法もある．

おわりに

Cyclic Executiveは，ハード・リアルタイム向けのコンパクトな動的アーキテクチャである．最後に紹介した`Handler`と`CyclicExe`のフレームワークを使用すればRTOSなしで，動的特性を予測できるシステムを構築することが可能である．さらに，すでにイベント・ループ型で実装してしまったプログラムを見直してCyclic Executiveへ移行することで動的特性を良くすることもできる．イベント駆動と時間駆動の使い分けにはメリハリを持たせる必要がある．この使い分けについては別の機会に取り上げる．

参考文献・URL

(1) G. D. Carlow, “*Architecture of the Space Shuttle Primary Avionics Software System*”, Comm. ACM, 27, 9, 1984, pp. 926-936.
(2) http://with.esm.co.jp/2004/uml-robocon2004/toppage.htm
(3) http://www.robo-one.com/
(4) http://pcweb.mycom.co.jp/articles/2004/02/05/robo-one/
(5) http://www.robocup.or.jp/
(6) たとえば，http://www.robos.co.jp/index.html
(7) たとえば，2002年優勝のRoboDragon，http://www.aichi-pu.ac.jp/ist/lab/narulab/indexE.html
(8) http://www.can.bosch.com/
(9) http://www.can.bosch.com/content/TT_CAN.html
(10) http://www.ttagroup.org/, http://www.ttchip.com/
(11) http://www.flexray.com/
(12) T. P. Baker, A. Shaw, “*The Cyclic Executive Model and Ada*,” Real-Time Systems Symposium, 1988.
(13) “*Software Considerations in Airborne Systems and Equipment Certification*,” Document RTCA/DO-178B, RTCA, Inc., Washington, D.C., December 1992.
(14) TimeWiz： http://www.co-nss.co.jp/products/realtime/timesys/timewiz.html, RapidRMA： http://www.tripac.com/html/prod-fact-rrm.html

ふじくら・としゆき　(株)豆蔵

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第㉗回 アセンブラを使いこなすための基礎知識とC言語からのアセンブラの使用方法（MASM編・その2）

大貫 広幸

今回は，前回のつづきとして実際のMASMによるVisual C++のサブルーチンの作成方法と，Visual C++がもつインライン・アセンブラについて説明します．

Visual C++のためのMASMによるアセンブラ・サブルーチンの作成方法

ここでは，Windows用の32ビットC/C++コンパイラであるVisual C++ 6.0と，32ビット用のアセンブラMASM Ver6.1を使用して，C言語のプログラムからアセンブラのサブルーチンを呼び出す方法について説明します．この方法は，Visual Studio .NET付属のVisual C++.NETとMASM Ver7でも同様です．

このC/C++言語のプログラムからのアセンブラのサブルーチンの呼び出しは，ルールさえわかってしまえばそれほど難しくはありません．

● C/C++言語のプログラム側での準備

まず，C/C++言語のプログラム側では，呼び出すアセンブラのサブルーチンをプロトタイプ宣言し，C/C++コンパイラにこれから使用するアセンブラのサブルーチンを登録します．

アセンブラのサブルーチンの登録は，プロトタイプ宣言で通常のC言語の関数と同じように宣言します．このとき呼び出す側の言語がC++言語だった場合は，アセンブラのサブルーチンはC言語の関数として，

```
extern "C" {
    // ここにアセンブラのサブルーチンを
    // C言語の関数として宣言する
}
```

と宣言します．このプロトタイプ宣言が済めば，通常のC言語の関数と同じ要領で，アセンブラのサブルーチンを呼び出すことができます．

また，アセンブラのプログラム側で定義されているデータ（変数）をC/C++言語のプログラム側からアクセスする場合は，このアセンブラのプログラム側で定義されているデータもC言語の外部変数としてexternで宣言しておきます．

Visual C++で使われる型とアセンブラMASMのデータの型は，表1のように対応しています．

表1 Visual C++の変数の型とアセンブラMASMのデータ型の対応

Visual C++のデータの型	MASM Ver 6.14, Ver 7.Xのデータの型	バイト数
unsigned char	BYTE, DB	1
unsigned short	WORD, DW	2
unsigned long	DWORD, DD	4
unsigned int	DWORD, DD	4
char	SBYTE	1
short	SWORD	2
long	SDWORD	4
int	SDWORD	4
float	REAL4	4
double	REAL8	8
ポインタ	DWORD,DD	4

● 呼び出し時の実引き数の受け渡し

C/C++言語のプログラム側から，アセンブラのサブルーチンを呼び出すとき，C/C++言語のプログラムからアセンブラのサブルーチンに実引き数を渡すことができます．この実引き数の個数と型は，プロトタイプ宣言時に決定されます．

Visual C++では，実引き数はすべてスタックにPUSHされた状態で渡されます．そして，整数やポインタの実引き数に対しては，型に関係なくつねに4バイト（32ビット）のサイズでスタックにPUSHされます．浮動小数点の値は，float型の値なら4バイト（32ビット），double型の値なら8バイト（64ビット）でスタックにPUSHしてきます．

実引き数がスタックにPUSHされる順番は，記述上から見て，右から左に向かって値がPUSHされます．たとえば，「func(aa, bb, cc)」の場合，まずccがPUSHされ，次にbb，そして最後にaaがPUSHされます．

アセンブラのサブルーチン側では，このスタック上の実引き数をレジスタEBPを使いアクセスします．そのための処理をアセンブラのサブルーチンでは，最初に，

```
    PUSH    EBP
    MOV     EBP, ESP
```

を実行します．先頭（左端）の実引き数は，[EBP+8]のアドレスにあります．

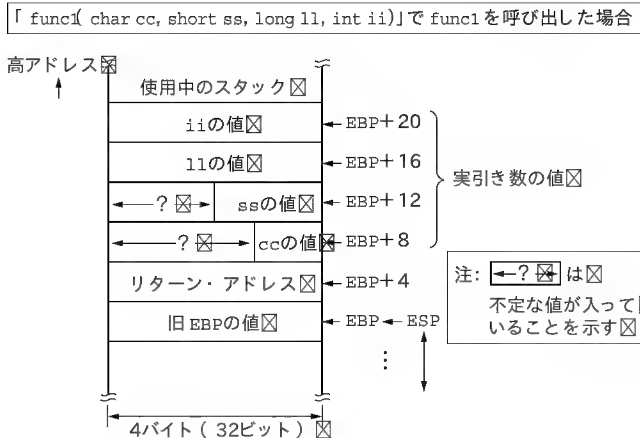


図1　Visual C++からC言語の関数を呼び出したときの実引き数

そして，サブルーチンからのリターンでは，

```
MOV     ESP, EBP
POP     EBP
RET
```

を実行します．

実引き数のスタックからの消去は，実引き数をスタックにPUSHした側，つまり呼び出した側で行います．

この実引き数の受け渡しの状況を，図で表すと図1のようになります．この呼び出し時の実引き数の受け渡しがアセンブラでプログラムできれば，アセンブラのプログラムからC言語の関数を呼び出すことも可能となります．

● アセンブラのサブルーチンからの戻り値

プロトタイプ宣言のとき戻り値をvoidとした場合は，アセンブラのサブルーチンは戻り値を返す必要はありません．しかし，プロトタイプ宣言で戻り値の型を指定した場合は，その型でアセンブラのサブルーチンは戻り値を返す必要があります．戻り値が整数値やポインタならCPUのレジスタAL，AX，EAXに戻り値を設定します．また，戻り値が浮動小数点の値ならFPUのST(0)に戻り値を設定して戻ります(表2)．

● プログラミング上の注意

ここでは，アセンブラでサブルーチンを作成する場合の注意点について解説します．

(1) アセンブラのサブルーチン側での注意

C言語側から呼び出されるアセンブラのサブルーチンやアクセスされる変数の名前には，必ずアンダ・スコア(_)を頭に付加します．次にアセンブラのサブルーチン側では，CPUのレジスタEAX，ECX，EDXの三つのレジスタは，退避することなく自由に使用できます．ほかのCPUのレジスタを使う場合は，スタックにレジスタの値を退避し，呼び出したルーチンに戻る前にレジスタの値を元の値に戻す必要があります．

FPUのレジスタは，レジスタ・スタックのみ使用できます．ただし，レジスタ・スタックは，呼び出したルーチンに戻るとき，最初にそのアセンブラのルーチンが呼ばれたときの状態に戻す必要があります．ただし，浮動小数点の戻り値がある場合のみST(0)が戻り値に使われます．

表2　アセンブラからVisual C++への戻り値の型と格納場所

戻り値	格納場所
unsigned char	レジスタAL
unsigned short	レジスタAX
unsigned long	レジスタEAX
unsigned int	レジスタEAX
char	レジスタAL
short	レジスタAX
long	レジスタEAX
int	レジスタEAX
float	FPUのST(0)
double	FPUのST(0)
ポインタ	レジスタEAX

また，MMXレジスタを使用した場合も必ず，FPUのレジスタスタックに戻し，アセンブラのルーチンが呼ばれたときの状態にする必要があります．XMMレジスタについては規定はありませんが，やはりアセンブラのルーチンが呼ばれたときの状態に戻しておいたほうがよいでしょう．

フラグは，演算より変化するステータス・フラグ以外は，変化させないようにします．たとえば，ストリング命令で使用するDFは，使用したら元の状態に戻してからリターンするようにします．

(2) アセンブラ側のルーチンからC言語側の変数のアクセスや関数を呼び出す場合の注意

C言語側の変数や関数の名前にはすべて頭にアンダ・スコア(_)が付きます．そのため，アセンブラ側のルーチンからアクセスするC言語の変数や呼び出す関数は，EXTERNディレクティブで宣言するとき，忘れずに変数名，関数名の頭にアンダ・スコアを付けるようにします．

● 実際のプログラム例

リスト1は，実際のCコンパイラVisual C++ 6.0とアセンブラMASM Ver 6.1でのプログラム例です．プログラムは，Windowsのコンソール・アプリケーションとして作成しました．

このプログラムでは，C言語のプログラムからアセンブラのサブルーチン呼び出しと，その逆のアセンブラのサブルーチンからC言語の関数呼び出し，そしてアセンブラのサブルーチンからのC言語の変数のアクセス，C言語のプログラムからアセンブラ側のデータのアクセスといった操作を行っています．図2(p.158)は，このリスト1のプログラムのコンパイルと実行結果です．

Visual C++のインライン・アセンブラの使い方

Visual C++ 6.0とVisual Studio.NETに付属のVisual C++ .NETには，インライン・アセンブラと呼ばれるC/C++言語

リスト 1　Visual C++と MASM の相互呼び出しのプログラム例

```
#ifndef __ASSUB_H
#define __ASSUB_H

#pragma pack( 1)

#ifdef  __cplusplus
extern "C" {
#endif

/* アセンブラ上のデータの宣言 */
extern char   asVar_cc;
extern short  asVar_ss;
extern long   asVar_ll;
extern float  asVar_ff;
extern double asVar_dd;
extern char   *asPtr_ch;

/* アセンブラ上のサブルーチンの宣言 */
extern char   asFunc1( char cc,short ss,long ll) ;
extern short  asFunc2( char cc,short ss,long ll) ;
extern long   asFunc3( char cc,short ss,long ll) ;
extern float  asFunc4( float ff,double dd) ;
extern double asFunc5( float ff,double dd) ;
extern char   *asFunc6( unsigned short x) ;

#ifdef  __cplusplus
}
#endif

#pragma pack( )

#endif /* __ASSUB_H */
```

MASMの構造体および共用体のパッキングした配置は，C/C++言語でいう「pack(1)」に相当する☒

アセンブラ側の変数やサブルーチンは，C++言語のときは「extern"C"」を指定し，C言語の変数や関数として宣言する☒

アセンブラ側の変数やサブルーチンは，C++言語のときは「extern"C"」を指定し，C言語の変数や関数として宣言する☒

（a）C/C++言語が参照するアセンブラ側のデータとサブルーチンを宣言しているインクルード・ファイル（ファイル名は asSub.h）

```
; C言語側の変数の宣言
extern _ccVar_cc:sbyte
extern _ccVar_ss:sword
extern _ccVar_ll:sdword
extern _ccVar_ff:real4
extern _ccVar_dd:real8
extern _ccPtr_ch:dword

; C言語側の関数の宣言
extern _ccFunc:near

; C言語のライブラリ sprintf 関数の宣言
extern _sprintf:near
```

（b）アセンブラが参照する C/C++言語側の変数と関数を宣言しているインクルード・ファイル（ファイル名は cFunc.inc）

```
/*
    C/C++言語とアセンブラ言語の相互呼び出し例
    << C言語でのプログラム例 >>
*/

#include <stddef.h>
#include <stdio.h>

#include "asSub.h"

/* アセンブラからアクセスされる変数 */
char   ccVar_cc=0;
short  ccVar_ss=0;
long   ccVar_ll=0;
float  ccVar_ff=0.0;
double ccVar_dd=0.0;
char   *ccPtr_ch=NULL;

/* アセンブラから呼び出される関数 */
long ccFunc( unsigned short x,long y)
{
    long retVal;
    retVal = x + y;
    return retVal;
}

/* メイン関数 */
int main( void)
{
    char   rvVar_cc;
    short  rvVar_ss;
    long   rvVar_ll;
    float  rvVar_ff;
    double rvVar_dd;
    char   *rvPtr_ch;

    ccVar_cc = 8;
    ccVar_ss = 16;
    ccVar_ll = 32;
    ccVar_ff = 1.2345;
    ccVar_dd = -9.87654321;
    ccPtr_ch = "owari";

    rvVar_cc = asFunc1( 11,12,14) ;
    printf( "asFunc1 : %d %hd %ld :
                %d¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_cc) ;
    rvVar_ss = asFunc2( 21,22,24) ;
    printf( "asFunc2 : %d %hd %ld :
               %hd¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_ss) ;
    rvVar_ll = asFunc3( 31,32,34) ;
    printf( "asFunc3 : %d %hd %ld :
              %ld¥¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_ll) ;
    rvVar_ff = asFunc4( 4.12345,-4.54321) ;
    printf( "asFunc4 : %f %f : %f¥n",asVar_ff,asVar_dd,
                                                   rvVar_ff) ;
    rvVar_dd = asFunc5( 5.12345,-5.54321) ;
    printf( "asFunc5 : %f %f : %f¥n",asVar_ff,asVar_dd,
                                                   rvVar_dd) ;
    rvPtr_ch = asFunc6( 1234) ;
    printf( "asFunc6 : %s : %s¥n",asPtr_ch,rvPtr_ch) ;

    return 0;
}
```

（d）C言語側のプログラム（ファイル名は cMain.c）

```
//
//    C/C++言語とアセンブラ言語の相互呼び出し例
//    << C++言語でのプログラム例 >>
//

#include <cstddef>
#include <cstdio>

#include "asSub.h"

// アセンブラからアクセスされる変数
extern "C" {
    char   ccVar_cc=0;
    short  ccVar_ss=0;
    long   ccVar_ll=0;
}
extern "C" float  ccVar_ff=0.0;
extern "C" double ccVar_dd=0.0;
extern "C" char   *ccPtr_ch=NULL;

// アセンブラから呼び出される関数
extern "C" long ccFunc( unsigned short x,long y)
{
    long retVal;
    retVal = x + y;
    return retVal;
}

// メイン関数
int main( )
{
    char   rvVar_cc;
    short  rvVar_ss;
    long   rvVar_ll;
    float  rvVar_ff;
    double rvVar_dd;
    char   *rvPtr_ch;

    ccVar_cc = 8;
    ccVar_ss = 16;
    ccVar_ll = 32;
    ccVar_ff = 1.2345;
    ccVar_dd = -9.87654321;
    ccPtr_ch = "owari";

    rvVar_cc = asFunc1( 11,12,14) ;
    printf( "asFunc1 : %d %hd %ld : %d¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_cc) ;
    rvVar_ss = asFunc2( 21,22,24) ;
    printf( "asFunc2 : %d %hd %ld : %hd¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_ss) ;
    rvVar_ll = asFunc3( 31,32,34) ;
    printf( "asFunc3 : %d %hd %ld : %ld¥n",asVar_cc,asVar_ss,asVar_ll,rvVar_ll) ;
    rvVar_ff = asFunc4( 4.12345,-4.54321) ;
    printf( "asFunc4 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_ff) ;
    rvVar_dd = asFunc5( 5.12345,-5.54321) ;
    printf( "asFunc5 : %f %f : %f¥n",asVar_ff,asVar_dd,rvVar_dd) ;
    rvPtr_ch = asFunc6( 1234) ;
    printf( "asFunc6 : %s : %s¥n",asPtr_ch,rvPtr_ch) ;

    return 0;
}
```

「extern"C"」は{}でまとめて定義することも，一つ一つ記述して定義することもできる☒

C++言語でアセンブラを使用する場合，アセンブラからアクセスされる変数や呼び出される関数は，「extern"C"」を指定し，C言語の変数や関数として定義する必要がある☒

（e）C++言語側のプログラム（ファイル名は cppMain.cpp）

リスト 1　Visual C++とMASMの相互呼び出しのプログラム例（つづき）

```
.586
.model flat
        page      60,132      ← MASMが生成するリスト・ファイルのプログラム・リストの1ページ分の行数と桁数を指定するディレクティブ
        title     C/C++言語とアセンブラ言語の相互呼び出し例      ← プログラム・リストに付けるタイトルを指定するディレクティブ
        subtitle  アセンブラ言語側のプログラム      ← プログラム・リストに付けるサブ・タイトルを指定するディレクティブ

.xlist
        include   cFunc.inc      ← ディスク上にある別のソース・ファイルを読み込むためのディレクティブ
.list
;-----------------------------------------------
      ← リスト・ファイルへのアセンブル結果の出力を制御するためのディレクティブ
        「.XLIST」が出力しない指定，「.LIST」が出力する指定となる
.data
; C/C++言語側からアクセスされるデータ
        public   _asVar_cc,_asVar_ss,_asVar_ll
        public   _asVar_ff,_asVar_dd,_asPtr_ch
_asVar_cc    sbyte     0
_asVar_ss    sword     0
_asVar_ll    sdword    0
_asVar_ff    real4     0.0
_asVar_dd    real8     0.0
_asPtr_ch    dword     0

; 整数値を文字列に sprintf 関数で変換するときに
; 使用するフォーマットとバッファ
strFmt   db      "[%d]",0
strBuf   db      80 dup(' '),0

        page
.code
;
; char asFunc1( char cc,short ss,long ll) ;
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2      ← スタック上の実引き数の値のアクセスは，このようにEBPからのオフセットを「=」で事前にシンボルとして定義しておくとアクセスしやすい
_asFunc1 proc    near      ← PROCで定義したラベルは，自動的にPUBLICとなる
        push    ebp
        mov     ebp,esp
;
        mov     al,arg_cc[ebp]      ← 実引き数はレジスタEBPを使用し，アクセスする
        mov     _asVar_cc,al
        mov     ax,arg_ss[ebp]
        mov     _asVar_ss,ax
        mov     eax,arg_ll[ebp]
        mov     _asVar_ll,eax
;
        mov     al,_ccVar_cc
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc1 endp
;
; short asFunc2( char cc,short ss,long ll) ;
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2
_asFunc2 proc    near
        push    ebp
        mov     ebp,esp
;
        mov     al,arg_cc[ebp]
        mov     _asVar_cc,al
        mov     ax,arg_ss[ebp]
        mov     _asVar_ss,ax
        mov     eax,arg_ll[ebp]
        mov     _asVar_ll,eax
;
        mov     ax,_ccVar_ss
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc2 endp
;
; long asFunc3( char cc,short ss,long ll) ;
arg_cc = 8
arg_ss = 8+4
arg_ll = 8+4*2
_asFunc3 proc    near
        push    ebp
        mov     ebp,esp
;
        mov     al,arg_cc[ebp]
        mov     _asVar_cc,al
        mov     ax,arg_ss[ebp]
        mov     _asVar_ss,ax
        mov     eax,arg_ll[ebp]
        mov     _asVar_ll,eax
;
        mov     eax,_ccVar_ll
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc3 endp
;
; float asFunc4( float ff,double dd) ;
arg_ff = 8
arg_dd = 8+4
_asFunc4 proc    near
        push    ebp
        mov     ebp,esp
;
        fld     real4 ptr arg_ff[ebp]
        fstp    _asVar_ff
        fld     real8 ptr arg_dd[ebp]
        fstp    _asVar_dd
;
        fld     _ccVar_ff
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc4 endp
;
; double asFunc5( float ff,double dd) ;
arg_ff = 8
arg_dd = 8+4
_asFunc5 proc    near
        push    ebp
        mov     ebp,esp
;
        fld     real4 ptr arg_ff[ebp]
        fstp    _asVar_ff
        fld     real8 ptr arg_dd[ebp]
        fstp    _asVar_dd
;
        fld     _ccVar_dd
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc5 endp
;
; char *asFunc6( unsigned short x) ;
arg_x = 8
_asFunc6 proc    near
        push    ebp
        mov     ebp,esp
;
        mov     eax,60000      ← C/C++言語側の関数ccFuncの呼び出し．C/C++言語の「ccFunc(x,6000)」に相当する呼び出し
        push    eax
        mov     ax,arg_x[ebp]      ← ccFuncへ渡す実引き数のスタックへのPUSH
        push    eax
        call    _ccFunc      ← C言語の関数ccFuncの呼び出し
        add     esp,4*2      ← スタック上の実引き数を消去する．POPを繰り返すよりESPに実引き数のサイズを加算したほうが速い
;
        push    eax
        push    offset flat:strFmt
        push    offset flat:strBuf
        call    _sprintf
        add     esp,4*3
        mov     eax,offset flat:strBuf
        mov     _asPtr_ch,eax      ← 「sprintf( strBuf,"[%d]",レジスタのEAXの値)」に相当する呼び出し．このようにするとアセンブラのプログラムからC言語で使われているライブラリ関数も使用できるようになる
;
        mov     eax,_ccPtr_ch
;
        mov     esp,ebp
        pop     ebp
        ret
_asFunc6 endp
;
end
```

（c）アセンブラ側のプログラム（ファイル名は asSub.asm）

図2　リスト1のプログラムのアセンブル，コンパイルと実行結果

```
D:¥WORK¥CqIfAsm¥LIST6>ml /c /coff /Cx /Fl asSub.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.
                                  All rights reserved.

 Assembling: asSub.asm

D:¥WORK¥CqIfAsm¥LIST6>cl cMain.c asSub.obj
Microsoft (R) 32-bit C/C++ Optimizing Compiler
                        Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998.
                                  All rights reserved.

cMain.c
Microsoft (R) Incremental Linker Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998.
                                  All rights reserved.

/out:cMain.exe
cMain.obj
asSub.obj

D:¥WORK¥CqIfAsm¥LIST6>cl cppMain.cpp asSub.obj
Microsoft (R) 32-bit C/C++ Optimizing Compiler
                        Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998.
                                  All rights reserved.

cppMain.cpp
Microsoft (R) Incremental Linker Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998.
                                  All rights reserved.

/out:cppMain.exe
cppMain.obj
asSub.obj

D:¥WORK¥CqIfAsm¥LIST6>cMain
asFunc1 : 11 12 14 : 8
asFunc2 : 21 22 24 : 16
asFunc3 : 31 32 34 : 32
asFunc4 : 4.123450 -4.543210 : 1.234500
asFunc5 : 5.123450 -5.543210 : -9.876543
asFunc6 : [61234] : owari

D:¥WORK¥CqIfAsm¥LIST6>cppMain
asFunc1 : 11 12 14 : 8
asFunc2 : 21 22 24 : 16
asFunc3 : 31 32 34 : 32
asFunc4 : 4.123450 -4.543210 : 1.234500
asFunc5 : 5.123450 -5.543210 : -9.876543
asFunc6 : [61234] : owari

D:¥WORK¥CqIfAsm¥LIST6>
```

アセンブラはMASM Ver6.14，C/C++コンパイラはVisual C++6.0のCL Ver12.00を使用

注：Visual C++.NETのアセンブラMASM Ver7.XXと，C/C++コンパイラCL Ver13.XXでもアセンブル，コンパイルそして実行が可能

ソース・プログラム内に，直接アセンブリ命令を記述する機能をもっています．このインライン・アセンブラは，CPUの命令は使いたいが，MASMを使いサブルーチンを作るほどではないという場合に便利です．

● __asmキーワード

Visual C++では，インライン・アセンブラは__asmキーワードを使って記述します．__asmは，単体で使用することでアセンブリ命令一つを記述することができます．また，__asm{　}を使うことで複数行のアセンブリ命令を記述することができます．

たとえば，

```
__asm   mov eax, verA
__asm   rol eax, 1
__asm   mov verB, eax
```

は，

```
__asm {
          mov eax, verA
          rol eax, 1
          mov verB, eax
    }
```

と記述することができます（リスト2）．このように__asmキーワードで記述された部分を中かっこ{　}のあるなしに関係なく「__asmブロック」と呼びます．

● __asmブロック内に記述できる命令

__asmブロック内では，CPUの命令セットと一部のディレクティブが記述できます．また，セミコロン（;）を使用したコメントもMASMと同様に記述することができます．

CPUの命令セットは，Visual C++ 6.0がMMXを含むPentiumの命令まで使用でき，Visual C++.NETではさらにSSE，SSE2命令も使用可能となっています．

オペランドに記述する式も，値やアドレスと評価できるものは記述可能です．オペランドで参照されるシンボルは，C/C++言語のソース・プログラムで使用されている名前をそのまま指定します．つまり，名前の頭にアンダ・スコア（_）は付加しません．

次に，ディレクティブの記述ですが，Visual C++のインライン・アセンブラはディレクティブをまったくといってよいほどサポートしていません．そのため，__asmブロックでデータを定義することはできません．唯一，__asmで使用できるディレクティブは，EVENとALIGNのみです．EVENとALIGNは，MASMのEVENとALIGNと同一のものです．インライン・アセンブラでEVENとALIGNを使用すると，飛んだアドレスの隙間はNOP命令で埋められます．

MASMにはなく，インライン・アセンブラでのみ存在する命令としては，「疑似命令_emit」があります．_emitは，.codeセグメントに1バイトの値を埋め込むものです．そのため，複数バイトの値を.codeセグメントに埋め込む場合は，複数回この_emitを使う必要があります．

● インライン・アセンブラのラベルとジャンプ

__asm {　}内にはジャンプ先となるラベルを記述することができます．ラベルは，MASMと同じでラベル名のすぐ後に

リスト 2　Visual C++での __asm の記述例

```
#include <stddef.h>
#include <stdio.h>

unsigned long varA;

unsigned long asmBlk1( long cnt)
{
    unsigned long varB;
    if(  cnt==0 )  return varA;
    if(  cnt>0 )  {
      __asm   mov   edx,varA
      __asm   mov   cl,byte ptr cnt
      __asm   rol   edx,cl     ; 左に CL ビット回転する
      __asm   mov   varB,edx
    }
    else { /* cnt<0 */
      cnt = -cnt;
      __asm   mov   edx,varA
      __asm   mov   cl,byte ptr cnt
      __asm   ror   edx,cl     ; 右に CL ビット回転する
      __asm   mov   varB,edx
    }
    return varB;
}

unsigned long asmBlk2( long cnt)
{
    unsigned long varB;
    if(  cnt==0 )  return varA;
    if(  cnt>0 )  {
      __asm mov edx,varA __asm mov cl,byte ptr cnt
      __asm rol edx,cl __asm mov varB,edx ; 左に CL ビット回転する
    }
    else { /* cnt<0 */
      cnt = -cnt;
      __asm mov edx,varA __asm mov cl,byte ptr cnt
      __asm ror edx,cl __asm mov varB,edx ; 右に CL ビット回転する
    }
    return varB;
}

unsigned long asmBlk3( long cnt)
{
    unsigned long varB;
    if(  cnt==0 )  return varA;
    if(  cnt>0 )  __asm {
      mov   edx,varA
      mov   cl,byte ptr cnt
      rol   edx,cl     ; 左に CL ビット回転する
      mov   varB,edx
    }
    else { /* cnt<0 */
      cnt = -cnt;
      __asm {
        mov   edx,varA
        mov   cl,byte ptr cnt
        ror   edx,cl     ; 右に CL ビット回転する
        mov   varB,edx
      }
    }
    return varB;
}

unsigned short *w4pakAdd(
  unsigned short *uspA,unsigned short *uspB
) {
    static unsigned short w4pak_x[4];
    __asm {
      mov    esi,uspA
      mov    ebx,uspB
      movq   mm0,qword ptr [esi]
      paddw  mm0,qword ptr [ebx]
      movq   w4pak_x,mm0
      emms
    }
    return w4pak_x;
}

#if _MSC_VER>=1300

float *f4pakAdd(
  float *fpA,float *fpB
) {
    static float f4pak_x[4];
    __asm {
      mov      ebx,fpA
      movups   xmm0,oword ptr [ebx]
      mov      ebx,fpB
      movups   xmm1,oword ptr [ebx]
      addps    xmm0,xmm1
      movups   f4pak_x,xmm0
    }
    return f4pak_x;
}

double *d4pakAdd(
  double *dpA,double *dpB
) {
    static double d2pak_x[2];
    __asm {
      mov      ebx,dpA
      movupd   xmm0,oword ptr [ebx]
      mov      ebx,dpB
      movupd   xmm1,oword ptr [ebx]
      addpd    xmm0,xmm1
      movupd   d2pak_x,xmm0
    }
    return d2pak_x;
}

#endif  /* _MSC_VER>=1300 */
```

インライン・アセンブラでは，グローバルな変数，ローカルな変数，仮引き数も，そのままの名前で記述することができる

関数asmBlk1，asmBlk2，asmBlk3は__asmの記述のしかたが異なるのみで同じ動作をする

asmBlk1：アセンブリ命令一つに一つの__asmを使用した例（一行に一つの__asmを記述）

コメントの記述はセミコロン（;）を使用する

asmBlk2：これもアセンブリ命令一つに一つの__asmを使用（一行に複数の__asmを記述）

asmBlk3：一つの__asmで複数のアセンブリ命令を記述する場合の例

Visual C++6.0，Visual C++.NETのCLではインライン・アセンブラでMMX命令が使用できる

Visual C++6.0ではSSE/SSE2命令が使用できないのでCLのバージョンを調べている

Visual C++.NETのCLではインライン・アセンブラでSSE/SSE2命令が使用できる

コロン（:）を付けて定義しています．この __asm { }内のラベルへは，__asm で記述された CPU の転送制御命令（ジャンプ命令など）のほか，C/C++ 言語の goto 文でもジャンプすることができます（リスト 3）．

● インライン・アセンブラと CPU のレジスタ

__asm で使用可能なレジスタは，EAX，EBX，ECX，EDX，ESI，EDI の六つのレジスタです．これ以外のレジスタは内容を変更しないように注意が必要です．またフラグは，演算より変化するステータス・フラグ以外は，変化させないようにします．

*　　　　　*

次回は，「アセンブラを使いこなすための基礎知識と C 言語からのアセンブラの使用方法」の gas 編を解説します．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

リスト 3　Visual C++と __asm でのラベルの使用

```
long gotoJump( long aa)
{
    if(  aa>=0 )  goto asmLb1;   ※1

    __asm {
      nop
      nop
      jmp   cLb1                 ※2
asmLb1:
      mov   ecx,aa
asmLb2:
      nop
      nop
      loop asmLb2
      nop
    }

    aa *= 2;
cLb1:
    return aa;
}
```

※1 C/C++言語のgoto文で__asmブロック内のラベルにジャンプすることができる

※2 __asmブロック内のジャンプ命令でC/C++言語のラベルへジャンプすることもできる

TMS320C6713搭載DSPスタータ・キットを使った

C++によるDSPオブジェクト指向プログラミング

第6回 FFTを利用するディジタル・フィルタ

◆三上 直樹

前回はFFTを利用して畳(たた)み込みを行うためのクラスを作成しました．今回は，このクラスを使って，FIRフィルタ(Finite Impulse Response Filter)のプログラムを作成します．

最初に作成するプログラムは，前回説明した重複加算法と重複保持法の動作を確認するためのものです．このプログラムはリアルタイム動作を行わないもので，Code Composer Studio(CCS)のグラフィック表示機能を使って信号の波形を確認します．その次に，DSKに搭載されているA-D/D-A変換用CODECを使い，信号の入出力をリアルタイムで行うようなディジタル・フィルタを重複保持法で実現します．

表1 FIRフィルタの設計時に与えたパラメータ

次数	12(係数の個数：13)	
	帯域1(通過域)	帯域2(阻止域)
下側帯域端周波数†	0.0	0.2
上側帯域端周波数†	0.1	0.5
利得	1	0
重み	1	2

†周波数は標本化周波数で正規化した値(標本化周波数を1とした値)で示している

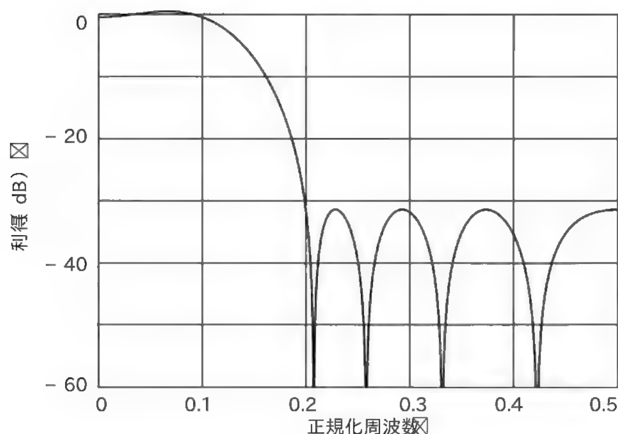


図1 作成するFIRフィルタの振幅特性

注1：この方法によるプログラムは文献(1)に付属するCD-ROMに収録されている．

注2：標本化周波数を1としたときの周波数．

FFTによるFIRフィルタのプログラム(非リアルタイム処理)

FFTを使ったFIRフィルタの処理をリアルタイム処理に適用する場合，データの入出力や移動などでいろいろと考慮しなければならない点があります．そこで，最初は動作の確認のため，入力信号があらかじめメモリ上に置かれているものとしてプログラムを作成します．入出力を含むリアルタイム処理を行うプログラムは次項で作成します．

作成するFIRフィルタは低域通過フィルタで，その係数はParks-McClellan法[注1]を使い，表1に示すパラメータで設計したものです．この表1で，周波数は標本化周波数を1としたときの正規化周波数で与えています．図1に設計したフィルタの振幅特性を示します．ここで作成するプログラムは重複加算法と重複保持法の動作を確かめる目的で作成するので，次数はあまり大きくしていません．そのため，低域通過フィルタとしての特性はあまり良いものではありません．

FIRフィルタを実現するために使う畳み込みには，前回作成したクラス`ConvolverFFT`を使います．そのクラスで使用するFFTの点数は64としました．

また，このプログラムでは実行結果が正しいことを確かめるため，同じフィルタ処理を時間領域で行う部分も含めています．

入力信号は，正規化周波数[注2]が0.053，0.237，0.382で振幅の異なる正弦波を重ね合わせたものを使いました．

なお，以下に示す二つのプログラムをビルドする際に使用するリンカ・コマンド・ファイルは，本連載第4回目のリスト7として説明したものと同じです．

● 重複加算法によるプログラム

リスト1に，重複加算法によるプログラム(`OverlapAdd.cpp`)を示します．このリストで網掛けの部分は，次の重複保持法によるプログラム(リスト2)とは異なる部分です．

以下の説明で使う記号は次のようになっています．

N：フィルタの係数の個数

M：使用するFFTの点数

入力信号および出力信号が格納される配列`x_n`，`y_n`と，時

リスト 1
重複加算法のプログラム（OverlapAdd.cpp）
網掛け部分は重複保持法のプログラムで置き換わる部分を示す

```
//-----------------------------------------------------------------------------
// 重複加算法による低域通過FIRフィルタのテスト・プログラム
//      FFTのデータ数: 64
// フィルタ設計のパラメータ
//      次数: 12
//      通過域:     0.0 -- 0.1 （正規化周波数）
//      阻止域:     0.2 -- 0.5 （正規化周波数）
//      重み: 通過域: 1, 阻止域: 2
//-----------------------------------------------------------------------------
#include "ConvolverFFT.cpp"
using namespace std;

const int nFIR    = 13;                 // フィルタ係数の数
const int nOrder  = nFIR - 1;           // FIRフィルタの次数
const int nFFT    = 64;                 // FFTのデータ数
const int L       = nFFT - nOrder;      //
const int Nb      =  5;                 // ブロックの数
const int Nall    = (Nb + 1)*L;         // フィルタ処理の対象となるデータの数
const float hm[nFIR] = {
    -2.20249554e-02,   -5.02358156e-02,   -2.77603826e-02,    3.21999472e-02,
     1.46234473e-01,    2.47866768e-01,    2.93655664e-01,    2.47866768e-01,
     1.46234473e-01,    3.21999472e-02,   -2.77603826e-02,   -5.02358156e-02,
    -2.20249554e-02};
const float M_2PI      = 2*3.14159265;
const float omegaL     = M_2PI*0.053f;     // 標本化角周波数を2πとする正規化角周波数
const float omegaH1    = M_2PI*0.237f;
const float omegaH2    = M_2PI*0.382f;

Array<float>  x_n(Nall), y_n(Nall), yD(Nall);

// メイン・プログラム
int main()
{
    Array<float> xnk(nFFT), ynk(nFFT), u(nFIR);
    ConvolverFFT Convolver(nFIR, nFFT, hm);

// フィルタ操作の対象となる信号の生成
    for (int n=0; n<Nall; n++)
        x_n[n] = sinf(omegaL*n) + 0.2f*sinf(omegaH1*n) + 0.3f*sinf(omegaH2*n);

// 重複加算法によるフィルタ処理の実行
    for (int n=0; n<nOrder; n++) y_n[n] = 0.0;
    for (int m=0; m<(Nall+1-nOrder)/L; m++)
    {
        for (int n=0; n<L; n++)              // 1ブロック分のデータを畳み込み用に領域に渡す
            xnk[n] = x_n[m*L+n];
        for (int n=L; n<nFFT; n++) xnk[n] = 0.0;
        Convolver.Execute(xnk, ynk)          // 畳み込みの実行
        for (int n=0; n<nOrder; n++)         // 重複部分の加算
            y_n[m*L+n] = y_n[m*L+n] + ynk[n];
        for (int n=nOrder; n<nFFT; n++)      // 加算を行わない部分
            y_n[m*L+n] = ynk[n];
    }

// 時間領域でのフィルタの実行
    for (int k=0; k<nFIR; k++) u[k] = 0;
    for (int n=0; n<Nall; n++)
    {
        u[0] = x_n[n];
        yD[n] = 0.0;
        for (int k=0; k<nFIR; k++)           // 出力信号の計算
            yD[n] = yD[n] + hm[k]*u[k];
        for (int k=nFFT-1; k>0; k--)         // データのシフト
            u[k] = u[k-1];
    }
}
```

間領域で行うフィルタ処理で使う配列yDは，前回作成した汎用1次元配列クラスArrayのグローバルなオブジェクトとして宣言しています．これらをmain関数内のローカルなオブジェクトとして宣言しても，プログラムの動作は変わりません．しかし，そのようにすると，CCSのグラフィック表示機能を使う場合に，アドレスの設定に手間がかかってしまいます．

main()関数では，まず作業領域として使う配列xnk，ynk，uが汎用1次元配列クラスArrayのオブジェクトとして宣言されています．また，FFTを使って畳み込みを行うためのクラスConvolverFFTのオブジェクトがConvolverという名前で宣言されています．

main()関数での処理は次のように進みます．

最初に，入力信号を発生させます．この信号は，周波数と振幅が違う三つの正弦波を重ね合わせたものです．次に，1ブロックごとの処理に入ります．最後に結果の比較のため，次の式（1）を使って時間領域でフィルタを実行します．

リスト 2　重複保持法のプログラム(OverlapSave.cpp)
リスト 1の網掛け部分を置き換える部分

```
// 先頭に nOrder 個の 0 を詰める
    Array<float> x_n0(Nall+nOrder);
    for (int n=0; n<Nall+nOrder; n++)
        x_n0[n] = (n<nOrder) ? 0.0 : x_n[n-nOrder];

// 重複保持法によるフィルタ処理の実行
    for (int m=0; m<(Nall+1-nOrder)/L; m++)
    {
        for (int n=0; n<nFFT; n++)
                     // 1 ブロック分のデータを畳み込み用に領域に渡す
            xnk[n] = x_n0[m*L+n];
        Convolver.Execute(xnk, ynk);        // 畳み込みの実行
        for (int n=0; n<L; n++)
                   // 先頭の nOrder 個のデータを除いて出力用配列に格納
            y_n[m*L+n] = ynk[n+nOrder];
    }
```

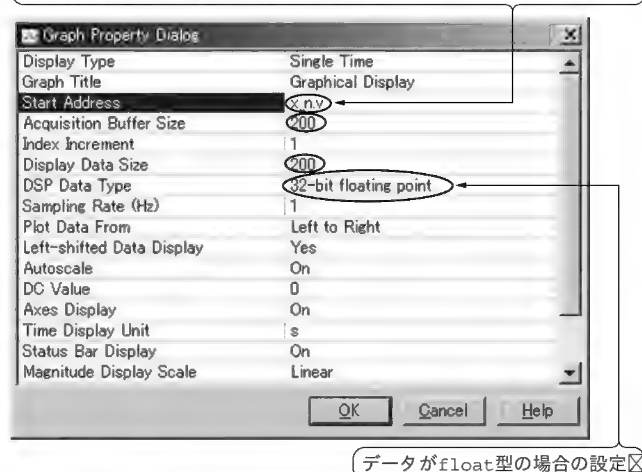


図2　信号の波形をグラフィックス表示する際の設定(円で囲んだ部分は変更した箇所を示す)

$$y[n]=\sum_{m=0}^{N-1} h[m]\cdot x[n-m] \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (1)$$

分割した1ブロックに対して FFT を使って畳み込みを実行する処理は，クラス ConvolerFFT のメンバ関数 Execute() が行います．

1ブロックごとの処理は次のように行います．1ブロックのデータ数は $M-N+1$になっているので，メンバ関数 Execute() の作業領域 xnk にデータを渡す際は，データの後に $N-1$個の0を追加します．次にメンバ関数 Execute() を実行し，その結果が格納される配列 ynk の中で，先頭の $N-1$個のデータは一つ前のブロックの，後の $N-1$個のデータと加え合わせて，出力データが格納される配列 y_n に出力します．残りの $M-N+1$個のデータはそのまま出力データが格納される配列 y_n に出力します．

● 重複保持法によるプログラム

リスト 2に重複保持法によるプログラム(OverlapSave.cpp) を示します．このリスト 2では，リスト 1の網掛け部分を置き換える箇所のみを示しています．

1ブロックごとの処理に入る前に，入力信号の先頭に，$N-1$個の0を追加します．

1ブロックごとの処理は次のように行います．1ブロックのデータをクラス ConvolerFFT のメンバ関数 Execute() の作業領域 xnk に渡します．この場合は重複加算法と違い，データ数は FFT の点数と同じなので，後に0を追加することは行いません．次に関数 Execute() を実行しその結果が格納される配列 ynk の中で，先頭の $N-1$個のデータは捨てて，残りの $M-N+1$個のデータを出力データが格納される配列 y_n に出力します．

● 実行結果

CCS は信号をグラフィック表示する機能を備えているので，これを使って実行結果を確認します．この機能は，信号の波形だけでなく，スペクトルなどの表示もできます．

この機能を使うためには，次のように操作します．

①メニュー・バーで[View]をクリックし，表示されるメニューから[Graph｜ Time/Frequency...]を選択する

②“ Graph Property Dialog”ウィンドウが表示されるので，必要に応じて各項目の設定を行う

実行結果は，波形とスペクトルで表示します．

波形の表示

波形を表示する場合は“ Graph Property Dialog”ウィンドウで図2のように設定しました．デフォルトの状態から変更した項目は円で囲んでいます[注3]．

図3に実行後の信号のようすを示します．実行結果はどちらのプログラムも同じになります．(a)は入力信号，(b)は出力信号です．ここで使用したフィルタは低域通過フィルタなので，出力信号からは高い周波数成分が取り除かれていることがわかります．

なお，比較のために時間領域で行った結果はここでは示しませんが，それを表示してみたところ，図3(b)に示す結果と同じになり，重複加算法および重複保持法によるプログラムが処理を正しく行っていることを確認することができました．

スペクトルの表示

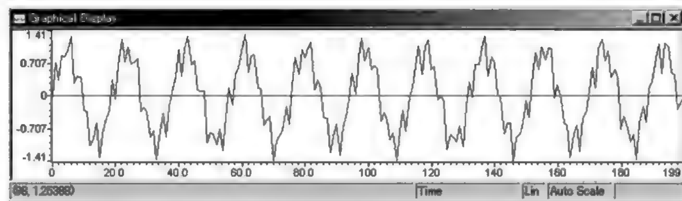
スペクトルを表示する場合は“ Graph Property Dialog”ウィ

注3：“ Start Address”の項目の指定において，16進数でアドレスを指定する場合は特に問題はない．しかし，配列などのシンボルでアドレスを指定する場合は次の二つの注意が必要になる．①ソース・プログラムを記述する場合に，表示させたいデータが格納される配列などはグローバルとして宣言する必要がある．②表示する領域が C++ 組み込みの配列の場合を除き，今回のように表示する領域の先頭アドレスがクラスのメンバであるポインタで指示される場合に，“ Release”版としてビルドすると，デフォルトでは詳細なデバッグ情報が生成されないので，ビルドのオプションの“ Generate Debug Info:”の項を“ No Debug”以外の状態に設定しておく必要がある．

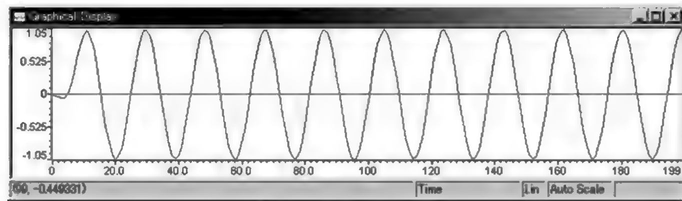
ンドウで図4のように設定しました．表示される値は，(a)ではFFTの絶対値，つまりリニア・スケールに，(b)ではそれをdB単位に変換した値，つまりログ・スケールになります．出力のスペクトルを表示する際の設定では次のような注意が必要です．

配列y_nに格納される出力信号のなかで，先頭からFIRフィルタの次数に相当する期間は過渡現象の部分に相当します．そのため，その部分を除いた部分に対応するスペクトルを表示したいので，“Start Address”には“y_n.v+0x10”という値を設定しています．

“FFT Windowing Function”の項目では，目的に応じて窓関数[注4]を選択します．デフォルトでは“Rectangle”，つまり方形窓になっていますが，この窓関数は信号の周期の整数倍がFFTの点数(図4の設定では 2^8=256)に一致しない場合は，一般的に望ましくありません．そこで，ここでは“Blackman”，つまりブラックマン窓を選択しています．



(a) 入力信号の波形

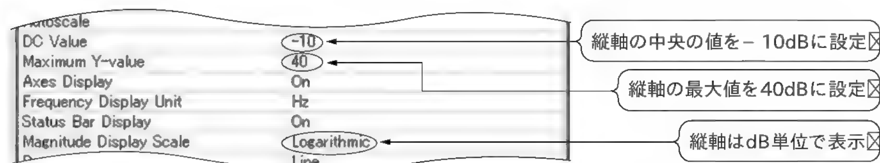


(b) 出力信号の波形

図3 重複保持法で行ったフィルタ処理の結果

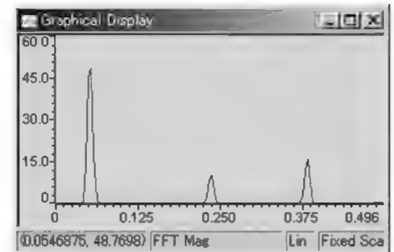


(a) 縦軸をリニア・スケールで表示する場合の設定

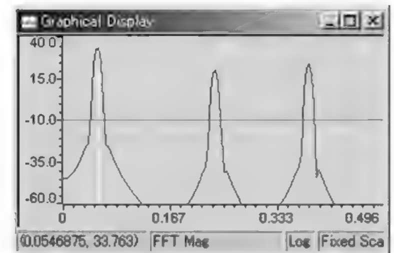


(b) 縦軸をdB単位で表示する場合の設定

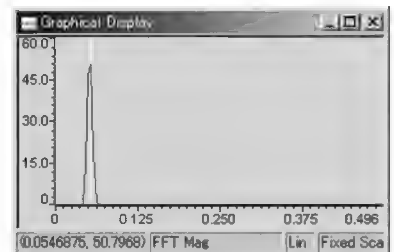
図4 信号のスペクトルをグラフィック表示する際の設定(円で囲んだ部分は変更した箇所を示す)



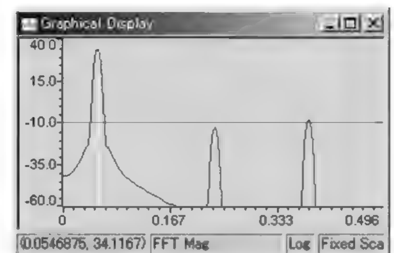
(a) 入力信号のスペクトル
(リニア・スケール表示)



(b) 入力信号のスペクトル(dB表示)



(c) 出力信号のスペクトル
(リニア・スケール表示)



(d) 出力信号のスペクトル(dB表示)

図5 入出力信号のスペクトル(リニア・スケール表示とdB表示)

注4：窓関数については次回に説明する．

縦軸をリニア・スケールにするかログ・スケールにするかの切り替えは，" Magnitude Display Scale"で選択します．ログ・スケールで表示する場合は，dB 単位になっています．またログ・スケールの場合に" DC Value"で設定する値は，縦方向の中央の値になります．

図5（p.163）に入力信号と出力信号のスペクトルを，リニア・スケール表示とログ・スケール（dB 表示）で示します．出力信号に含まれる周波数0.237，0.382の成分はフィルタで大きく減衰させられるので，リニア・スケールで表示するとどの程度まで減衰しているのかがよくわかりません．しかし，ログ・スケールで表示すると，周波数0.237，0.382の成分はともに30dB以上減衰していることがわかります．したがって，表示されたスペクトルからも，低域通過フィルタによる処理が期待したとおりに行われていることが確認できます．

表2 FIRフィルタの設計時に与えたパラメータ

次数	96（係数の個数：97）		
標本化周波数（kHz）	48		
	帯域1（阻止域）	帯域2（通過域）	帯域3（阻止域）
下側帯域端周波数（kHz）	0.1	2.0	4.0
上側帯域端周波数（kHz）	1.0	3.0	24.0
利得	0	1	0
重み	1	1	1

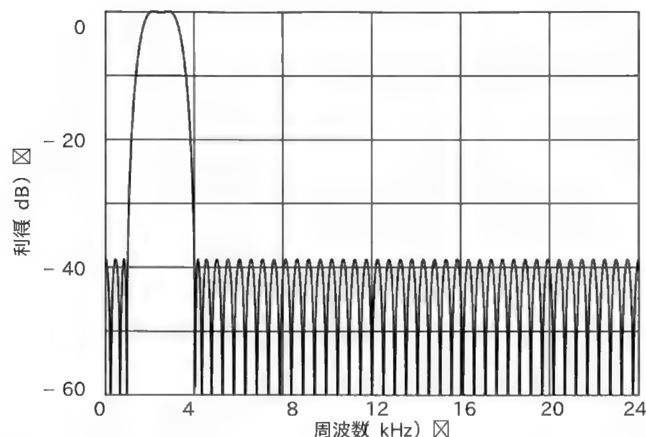


図6 作成するFIRフィルタの振幅特性

FFTによるFIRフィルタのプログラム（リアルタイム処理）

今度は，DSKボードに搭載されているA-D/D-A変換用のCODECを使って信号の入出力をリアルタイムで行うようなディジタル・フィルタを作成します．ここではプログラムを作りやすい重複保持法を使うことにします．また，リアルタイム動作をさせる場合に，データのバッファが必要になります．そこで，ここではバッファとして通常の配列を使う方法とキューを使う方法の二つの方法でプログラムを作成します．

作成するFIRフィルタは帯域通過フィルタで，その係数はParks-McClellan法[注5]を使い，表2に示すパラメータで設計したものです．図6に設計したフィルタの振幅特性を示します．使用するFFTの点数は256とします．

入力はDSKボードに搭載されているTLV320AIC23の左チャネルからの信号を使います．この1チャネルの入力信号に対して，FFTを使って実行したフィルタ処理の結果を左チャネルへ，通常の時間領域で行ったフィルタ処理の結果を右チャネルへ出力するというプログラムを作成します．

ところで，この連載で使っているC6713DSKのCCSにはDSP/BIOSという一種のリアルタイムOSも含まれています．したがって，ある程度以上の高度なマルチタスク・システムを構築する際は，このDSP/BIOSを使ったほうがシステムの開発を効率よく行うことができます．しかし，その使いかたを説明するとかなりのページ数を使うことになります．また，読者の中でリアルタイムOSが初めてという方にとっては，それを理解するだけでも大きな障壁になります．一方，ここで作る程度

注5：この方法によるプログラムは文献（1）に付属するCD-ROMに収録されている．

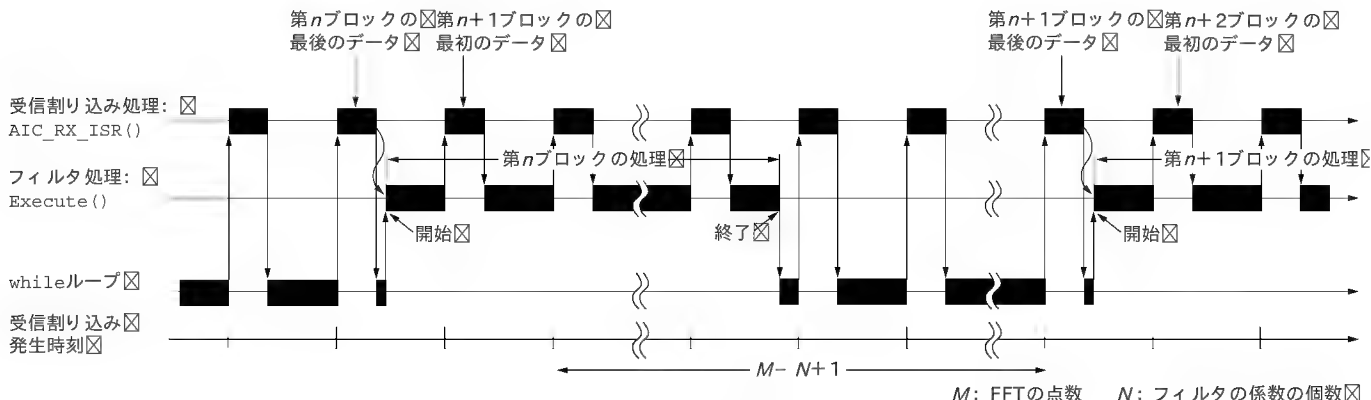


図7 FFTによるフィルタ処理をリアルタイム処理として行う際の全体の処理の流れ

の小規模なシステムでは，特にDSP/BIOSを使わなくてもプログラムの記述に不便はありません．そこで，ここではDSP/BIOSを使用しないでプログラムを作成します．なお，DSP/BIOSに興味のある読者は文献(2)などを参照してください．

● バッファとして通常の配列を使う方法

最初に作成するフィルタは，入力信号および出力信号のためのバッファとして，通常の配列を使って実現します．

▶ 本体のプログラム

本体のプログラムをリスト3(OverlapSaveArray.cpp)に示します．なお，このリストで網掛けした部分は次のバッファとしてキューを使う方法のプログラムで変更される部分を示しています．プログラム全体の処理のタイミングを図7に示します．

リスト3
通常の配列を使って重複保持法によりFIRフィルタの処理を実行するプログラムの中心部分
(OverlapSaveArray.cpp)
網掛け部分は重複保持法のプログラムで置き換わる部分

```
//---------------------------------------------------------------------------
// 重複保持法によるFIRフィルタと時間領域におけるFIRフィルタ
//   重複保持法:        通常の配列を使用
//   時間領域のFIRフィルタ: 転置形構成
//   FFTの点数: 256
//   フィルタ係数の数:   97
//---------------------------------------------------------------------------
#include "AIC23_Intr.hpp"
#include "OLS_Array.cpp"
#include "FIR_class.cpp"

const int N1   =  97;                           // フィルタ係数の数
const int nFFT = 256;                           // FFTの点数
const int N21p = nFFT - N1 + 1;
// 割り込み設定のためのデータ
const AIC23_Intr::IntrConfig IntrCfTbl[] =
                    { {IRQ_EVT_RINT1, 11},      // McBSP1受信割り込み用
                      {0, 0} };                 // データのターミネータ
AIC23_Intr codec(IntrCfTbl);

const float hm[N1] = {
    8.78230012e-04,    -4.18444706e-03,    2.01228923e-03,    2.17409546e-03,
    1.11062259e-03,    -2.29668333e-04,   -1.73482950e-03,   -3.36188084e-03,

                                          (省略)

    1.11062259e-03,     2.17409546e-03,    2.01228923e-03,   -4.18444706e-03,
    8.78230012e-04};

volatile bool runFilter;          // runFilterがtrueの場合にフィルタ処理の実行を行う

OLS_Array FIR_FFT(N1, nFFT, hm); // 重複保持法によるフィルタ処理のオブジェクト
FIR_Transpose FIR(N1-1);         // 時間領域でのFIRフィルタ処理を行うクラスの宣言(第3回目参照)

int main()
{
    short dummy[2];

    runFilter = false;           // 重複保持法によるフィルタ処理の禁止
    codec.Read(dummy);           // オーバラン・エラー・フラグをクリアするためのダミー読み出し

    IRQ_globalEnable();          // グローバル割り込みの許可

    while (1)
        if (runFilter)
        {
            FIR_FFT.Execute();
            runFilter = false;
        }
}

// McBSP1受信割り込みに対するルーチン
interrupt void AIC_RX_ISR()
{
    float ch0, ch1;
    static int count = 0;        // 入力されたデータ数のカウンタ

    codec.Read(ch0, ch1);        // A-D変換器からの入力，ch1は使用しない
    FIR_FFT.rxPut(ch0, count);   // 入力信号を入力バッファへ格納
    ch1 = FIR.Execute(hm, ch0);  // 時間領域のFIRフィルタ処理の結果をch1へ
    ch0 = FIR_FFT.txGet(count);  // 重複保持法によるフィルタ処理の結果をch0へ
    codec.Write(ch0, ch1);       // D-A変換器への出力

    if (++count >= N21p)
    {
        count = 0;
        runFilter = true;        // 重複保持法によるフィルタ処理の許可
    }
}
```

● インクルード・ファイル

今回のプログラムでは，DSKボードに搭載されているTLV320AIC23に接続されているMcBSP1の受信割り込みを使ってアナログ信号の入出力を行います．そのためにクラスAIC23_Intrを使うので，“AIC23_Intr.hpp”をインクルードします．次のインクルード・ファイルは“OLS_Array.cpp”で，これは基底クラスConvolverFFTの派生クラスOLS_Arrayのプログラムが書かれています．この内容についてはこの後に説明します．3番目のインクルード・ファイル“FIR_class.cpp”にはFIRフィルタ用のクラスのプログラムが書かれています．

● グローバルの宣言など

グローバル定数として，フィルタの係数の数(N1)，使用するFFTの点数(nFFT)，および出力する際の1ブロックのデータ数(N21p)の値を定義します．このプログラムで使用するFFTの点数は256とします．

クラスAIC23_Intrに関する宣言は次の二つです．一つはAIC23_Intrのメンバの一つである構造体IntrConfigのオブジェクトであるIntrCfTbl[]の宣言で，ここではMcBSP1の受信割り込みをCPU割り込みのINT11に割り当てるためのデータを設定しています．この構造体のオブジェクトは次の行で，クラスAIC23_Intrのオブジェクトであるcodecの宣言の際に引き数として使われます．

フィルタの係数hm[]は誌面のつごう上，大部分を省略しています．この係数の値についてはInterGiga No.33に収録するソース・リストを見てください．

volatile bool型の変数runFilterは，フィルタ処理を行うかどうかを決めるフラグとして使います．この変数は必ずvolatileとして宣言する必要があります．

FFTを使ったフィルタ処理に対応するクラスとしては，OLS_Arrayを使い，そのオブジェクトとしてFIR_FFTを宣言しています．

このプログラムでは，時間領域で処理を行うFIRフィルタも同時に走らせますが，そのフィルタは本連載第3回目に作成したFIRフィルタのためのクラスFIR_Transposeを使いました．このフィルタの構造は転置形(transposed form)になっています．

● main()関数

main()関数では，いくつかの初期化処理が終了した後，McBSPがオーバラン・エラーを発生している可能性があります．そのため，

```
codec.Read(dummy);
```

により，データを読み出して，McBSPのオーバラン・エラー・フラグをクリアします．

その後，CSL(Chip Support Library)[3]のAPI関数として提供されているIRQ_globalEnable()で，グローバル割り込みの許可を行った後，while文によるアイドル状態に入ります．

アイドル状態では，フラグとして使っている変数runFilterをつねに調べており，この値がtrueになればフィルタ処理に移行し，フィルタ処理が終了した変数runFilterをfalseに設定します．なお，変数runFilterがtrueに設定されるのは，受信割り込み処理AIC_RX_ISR()の中で入力信号が処理に必要な個数だけ読み込まれた場合です．

● 受信割り込み処理

受信割り込み処理はAIC_RX_ISR()で行います．AIC_RX_ISR()の先頭に付いているキーワードinterruptは，この処理が割り込み処理であることを指定します．

int型の変数countは入力データの数をカウントするためのものです．この値はAIC_RX_ISR()の処理が終了後，再びこの関数が呼び出された場合に，以前の値を保持されている必要があります．そのため，static変数として宣言しています．

信号の入出力はクラスAIC23_Intrのメンバ関数であるRead()とWrite()を使っています．入力した信号は左チャネルのみ使います．この入力信号を，メンバ関数rxPut()でFFTを使うフィルタの入力バッファに格納し，さらに同じ信号を時間領域のFIRフィルタを実行する関数FIR()に渡します．

次に，FFTを使うフィルタの出力バッファからメンバ関数txGet()で取り出した信号を左チャネルに，関数FIR()の処理結果を右チャネルに出力します．

最後に，処理を行ったデータ数が$M-N+1$(M: FFTの点数，N: フィルタの係数の個数)に達したかどうかをチェックします．その結果データ数が$M-N+1$に達していたら，データ数をカウントするcountをクリアし，フラグとして使っている変数runFilterをtrueにセットします．これにより，while文の中で，フィルタ処理が実行されるようになります．

▶OLS_Arrayのプログラム

バッファとして通常の配列を使う方法では，クラスConvolverFFTの派生クラスOLS_Arrayを使います．これをリスト4(OLS_Array.cpp)に示します．このクラスでは，入力信号および出力信号のためのバッファと作業領域との間のデータの転送と，クラスConvolverFFTのメンバ関数Execute()による畳み込みの計算を行います．

データの転送は高速に行う必要があるため，C6713のEDMA(Enhanced Direct Memory Access)を使います．そのため，CSL[3]のDATモジュールに含まれるAPI関数を使います．インクルード・ファイルcsl_dat.hはそのために必要なものです．

クラスOLS_Arrayのコンストラクタは，メンバ初期設定の機能(“:”以下の部分)を使って基底クラスのコンストラクタへ必要なデータを送り，さらに非公開メンバであるクラスArrayのオブジェクトrxBuf，txBufのコンストラクタにサイズを決めるデータを送り，コンストラクタを起動します．メンバ初期設定の後，関数DAT_open()でDATモジュールの関数を使うための初期化を行います．

デストラクタでは，DATモジュールのクローズを行います．

リスト 4
通常の配列を使って重複保持法によりFIRフィルタの処理を実行するプログラムで使用する ConvolverFFTの派生クラス(OLS_Array.cpp)

```
//-----------------------------------------------------------------
// 重複保持法で使うクラス ConvolverFFT の派生クラス
//  通常の配列を入出力バッファとして使用
//  作成者:  三上直樹, 2004
//-----------------------------------------------------------------
#ifndef MK_OLS_Array

#include "ConvolverFFT.cpp"
#include <csl_dat.h>    // CSL の DAT モジュールで使用

// 派生クラス OLS_Array の宣言部
class OLS_Array : public ConvolverFFT
{
private:
    Array<float> rxBuf, txBuf;
public:
    OLS_Array(const int nCoefs, const int nFFT, const float hm[])
              : ConvolverFFT(nCoefs, nFFT, hm),
                rxBuf(nFFT, 0.0), txBuf(nFFT-nCoefs+1, 0.0)
              { DAT_open(DAT_CHAANY, DAT_PRI_LOW, 0); }
    ~OLS_Array() { DAT_close(); }
    void Execute();
    void rxPut(float x, int n) { rxBuf[n+N-1] = x; }
    float txGet(int n) { return txBuf[n]; }
};

// 派生クラス OLS_Array の定義部
// 1 ブロックのデータに対する直線畳み込みの実行
// データのコピーは CSL の DAT モジュールを使用
void OLS_Array::Execute()
{
    int xferID;

// 前のブロックのフィルタ処理の結果を出力バッファへ転送
    xferID = DAT_copy((float*)tmp+N-1, txBuf, sizeof(float)*N21p);
    DAT_wait(xferID);
// 現在のブロックの入力信号を作業領域に
    xferID = DAT_copy(rxBuf, tmp, sizeof(float)*M);
    DAT_wait(xferID);
// 入力の重複部分を次のブロックの処理のため格納
    xferID = DAT_copy((float*)rxBuf+N21p, rxBuf, sizeof(float)*(N-1));

    ConvolverFFT::Execute(tmp, tmp);     // FFT による畳み込みの計算
}

#define MK_OLS_Array
#endif
```

rxPut()は入力信号を入力バッファ rxBufに格納するためのメンバ関数です．このとき，入力バッファの前の部分は，前のブロックのデータで重複して用いる部分がすでに格納されているので，その部分の次から格納していくようにしています．つまり，n番目の入力信号が入力バッファの$n+N-1$（N：フィルタの係数の個数）番目に格納されます．このようすを図8に示します．

txGet()は出力バッファ txBufから出力信号を取り出すためのメンバ関数です．

メンバ関数 Execute()の処理では，畳み込みの計算を行う前にデータの転送を行います．そのようすを図9に示します．このデータ転送は高速に行う必要があるため，関数DAT_copy()によりEDMAを使って行っています．その次の行の関数DAT_wait()は，DAT_copy()による転送が終了するまで待つための関数です．

最初に，一つ前のブロックでのフィルタ処理の結果を出力バッファ txBufへ転送します．このとき，先頭の$N-1$個のデータは不要なので，図9(a)に示すように配列の添え字のN-

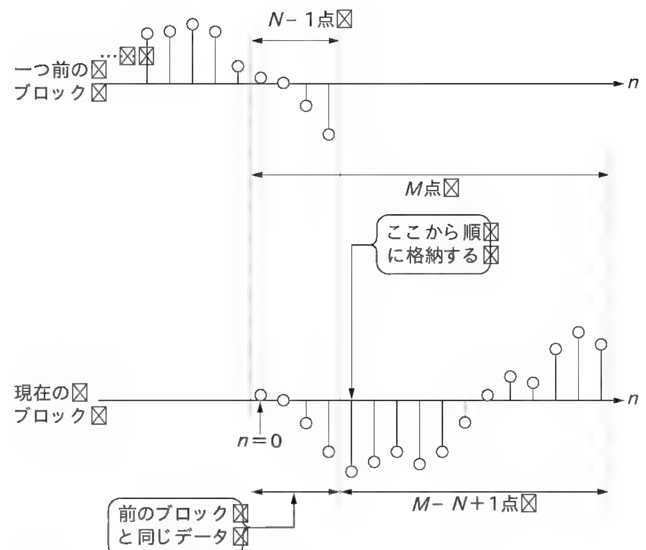


図8 重複保持法で入力信号を入力バッファに格納するようす（Nはフィルタ係数の個数を，Mは使用するFFTの点数を示す）

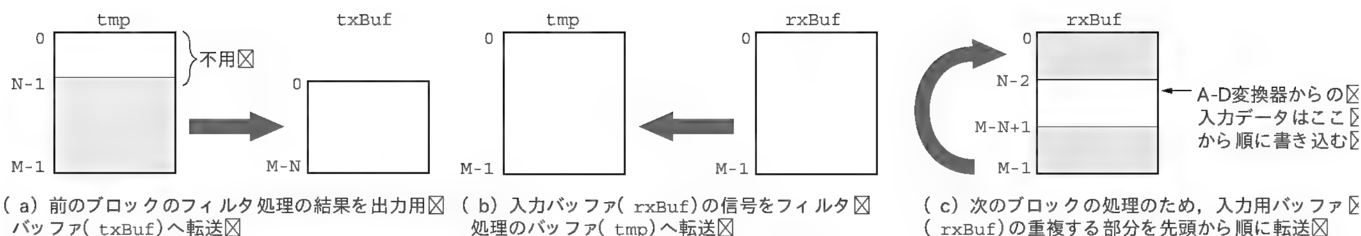


図9 クラスOLS_Arrayのメンバ関数Execute()の内部で行われるデータ転送のようす

1～M-1に対応する$M-N+1$個のデータを出力バッファtxBufの先頭から順に格納していきます．

次に，入力バッファrxBufのデータを，クラスConvolverFFTのメンバ関数Execute()で使う作業領域tmpに転送します．このときは，図9(b)に示すように，入力バッファrxBufのすべてのデータをtmpに格納します．

その後，入力バッファのデータの中で重複して用いる部分〔rxBuf後部の$N-1$個のデータで，図9(c)に示すように配列の添え字のM-N+1～M-1に対応するデータ〕を，次のブロックでの処理のため，入力バッファrxBufの先頭から順に転送します．

この転送の際にアクセスされる領域rxBufは次に行うFFTによる畳み込みの計算では使わないので，関数DAT_wait()で転送が終了するまでは待つ必要はありません．そこで，転送の終了を待たずに，クラスConvolverFFTのメンバ関数Execute()で畳み込みの計算を行います．

▶ プログラムのビルド時に使うファイル

ここまで説明してきたプログラムをビルドする際に使う，リセット/割り込みベクタを記述したファイルとリンカ・コマンド・ファイルについて説明します．

このプログラムでは割り込みとしてCPU割り込みのINT11を使うので，リセット/割り込みベクタを記述したアセンブリ言語によるファイルはリスト5のようになります．

このプログラムはライブラリとして，本連載第4回目で説明したFFTのライブラリを使います．そこで，リンカ・コマンド・ファイルは第4回目のリスト7と同じものを使います．リストは同じなので省略します．

● バッファとしてキューを使う方法

FFTを使ったフィルタで，最初に説明した入出力信号のためのバッファとして通常の配列を使って作成したフィルタの場合，出力信号は2ブロック分遅れることになります．この遅れをもっと少なくすることは，通常の配列を使った方法でも可能ですが，プログラムを作ることがやっかいになります．そこで，出力のバッファとしてキュー(queue)を使ったフィルタを作成します．入力のバッファには特にキューを使わなくてもよいのですが，ここで作るプログラムでは入力バッファもキューを使

リスト5 アセンブリ言語によるリセット・ベクタおよび割り込みベクタに関する記述(vects_Reset_INT11.asm)

```
;************************************************
;        リセット・ベクタおよび割り込みベクタ            *
;************************************************
        .sect "vectors"
        .ref  _c_int00           ; C/C++ のエントリ・ポイントのシンボル
        .ref  _AIC_RX_ISR__Fv    ; McBSP1の受信割り込みに対する割り込みサービス・ルーチンのシンボル

; リセット・ベクタ
RESET:
        MVKL  _c_int00,A0    ; A0に，_c_int00に対応するアドレスをロード
        MVKH  _c_int00,A0
        B     A0   ; A0で示されるアドレス( _c_int00) へ分岐
        NOP
        NOP
        NOP
        NOP
        NOP

        .space    0x20*10   ; 0x140バイトの領域を予約

; McBSP1の受信割り込みに対するベクタ
INT11:                       ; 先頭アドレス = 0x00000160
        STW   A0,*B15--[1]   ; A0をスタックへプッシュ
        MVKL  _AIC_RX_ISR__Fv,A0 ; A0に，_AIC_RX_ISR__Fvに対応するアドレスをロード
        MVKH  _AIC_RX_ISR__Fv,A0
        B     A0             ; McBSP1の受信割り込みに対する割り込みサービス・ルーチンへの分岐
        LDW   *++B15[1],A0   ; スタックからA0をポップ
        NOP
        NOP
        NOP2
```

リスト 6　キューを実現するクラス(MyExQueue.hpp)

```
//------------------------------------------------------
// リング・バッファによる拡張されたキューのためのテンプレート・クラス
// (C) MIKAMI, Naoki: 2004
//------------------------------------------------------
#ifndef MK_MyExQueue

#include "MyArray.hpp"

template <class T> class ExQueue
{
private:
    Array<T> rBuf;
    const int size;
    volatile int head, head2, tail;
public:
    ExQueue(int n) : size(n), rBuf(n+1) { Clear(); }
    inline bool Put(const T val);
    inline bool Get(T &val);
    inline bool GetCopy(T &val);
    void Clear() { head = head2 = tail = 0; }
};

// キューの最後部にデータを挿入
template <class T> inline bool ExQueue<T>::Put(const T val)
{
    int tmp = ((tail+1) > size) ? 0 : tail+1;
    if (tmp!=head)
    {
        rBuf[tail] = val;
        tail = tmp;
        return true;
    }
    else
        return false;
}

// キューの先頭からデータを取り出す
template <class T> inline bool ExQueue<T>::Get(T &val)
{
    int tmp = ((head+1) > size) ? 0 : head+1;
    if (head!=tail)
    {
        val = rBuf[head];
        head2 = head = tmp;
        return true;
    }
    else
    {
        val = (T)0;
        return false;
    }
}

// キューの先頭からデータをコピーする(データはそのまま)
template <class T> inline bool ExQueue<T>::GetCopy(T &val)
{
    int tmp = ((head2+1) > size) ? 0 : head2+1;
    if (head2!=tail)
    {
        val = rBuf[head2];
        head2 = tmp;
        return true;
    }
    else
    {
        val = (T)0;
        return false;
    }
}

#define MK_MyExQueue
#endif
```

いました．

なお，このプログラムをビルドする際に使用する，リセット/割り込みベクタを記述したファイルとリンカ・コマンド・ファイルは「バッファとして通常の配列を使う方法」の場合と同じものを使います．

▶ キューのプログラム

キューは待ち行列またはFIFO(first-in first-out)と呼ばれる場合もあります．このキューのプログラムをリスト6(MyExQueue.hpp)に示します．ここではキューをテンプレート・クラス(template class)として作成しました．したがって，このクラスは扱うデータの型に依存しないため，汎用性があります．

キューに対する基本的な操作としては，初期化，データの追加，データの取り出しです．ここでは重複保持法を実現することを容易にするため，キューの機能を拡張しました．つまり，データを取り出し，さらにそのデータは再び取り出すことができるようにするという操作も付け加えました．

リスト6では，リング・バッファを使ってキューを実現しています．その構造を図10に示します．データを追加する位置を示す添え字はtail，取り出す位置を示す添え字はheadです．そのほかにリスト6では，head2という添え字を使いますが，これはデータを取り出す際にそのデータは再び取り出すことができるようにするという操作で使う添え字です．

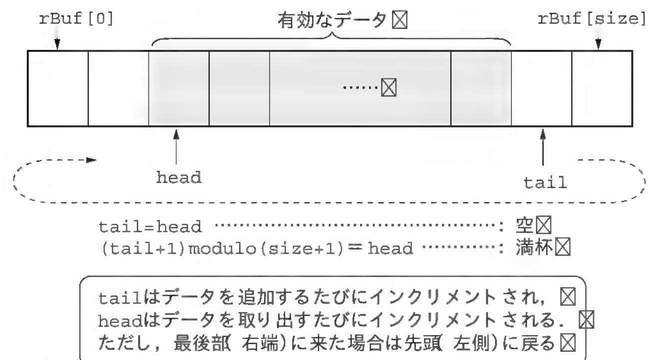


図10　リング・バッファを使ったキューの構造

なお，この三つの添え字tail，head，head2はvolatileとして宣言する必要があります．

このキューは，その中に入っている要素の数を表すカウンタを持っていません．そのため，size個までのデータを入れられるようにするため，キュー内部のバッファのサイズはsize+1としています．このようにすると，head=tailの場合にキューは空であることを検出できます．また，tail+1>sizeのときは0=headの場合，それ以外はtail+1=headの場合にキューは満杯であることが検出できます[注6]．

注6：キューが満杯になっている条件は(tail+1)modulo(size+1)=headと表現することもできる．

表3　本体のプログラムがリスト 3と異なっている箇所

記述されている箇所	リスト 3	バッファとしてキューを使う方法
インクルード・ファイル	#include "OLS_Array.cpp"	#include "OLS_Queue.cpp"
グローバル宣言	OLS_Array FIR_FFT(N1, nFFT, hm);	OLS_Queue FIR_FFT(N1, nFFT, hm);
受信割り込み処理	FIR_FFT.rxPut(ch0, count);	FIR_FFT.rxPut(ch0);
受信割り込み処理	ch0 = FIR_FFT.txGet(count);	ch0 = FIR_FFT.txGet();

リスト 7　キューを使って重複保持法により FIRフィルタの処理を実行するプログラムで使用する ConvolverFFT の派生クラス(OLS_Queue.cpp)

```
//-------------------------------------------------------------------
// 重複保持法で使うクラス ConvolverFFT の派生クラス
//  キューを入出力バッファとして使用
//  作成者:  三上直樹, 2004
//-------------------------------------------------------------------
#ifndef MK_OLS_Queue

#include "ConvolverFFT.cpp"
#include "MyExQueue.hpp"

// 派生クラス OLS_Queue の宣言部
class OLS_Queue : public ConvolverFFT
{
private:
    ExQueue<float> rxQueue; // 入力バッファ用キュー
    ExQueue<float> txQueue; // 出力バッファ用キュー
    void IntrGEnable() { CSR |= 0x1; }          // グローバル割り込み許可
    void IntrGDisable() { CSR &= 0xFFFFFFFE; } // グローバル割り込み禁止
public:
    OLS_Queue(const int nCoefs, const int nFFT, const float hm[]);
    void Execute();
    void rxPut(const float x) { rxQueue.Put(x); }
    float txGet() { float x; txQueue.Get(x); return x; }
};

// 派生クラス OLS_Queue の定義部
// コンストラクタ
OLS_Queue::OLS_Queue(const int nCoefs, const int nFFT, const float hm[])
         : ConvolverFFT(nCoefs, nFFT, hm), rxQueue(nFFT), txQueue(nFFT-nCoefs+1)
{
    for (int n=0; n<N-1; n++) rxQueue.Put(0.0);
}

// 1 ブロックのデータに対する直線畳み込みの実行
void OLS_Queue::Execute()
{
    for (int n=0; n<N21p; n++)        // 入力用キューからデータを取り出す
    {
        IntrGDisable();
        rxQueue.Get(tmp[n]);
        IntrGEnable();
    }
    for (int n=N21p; n<M; n++)        // 重複部分のデータに対応
    {
        IntrGDisable();
        rxQueue.GetCopy(tmp[n]);      // 入力用キューからデータをコピーする
        IntrGEnable();
    }

    ConvolverFFT::Execute(tmp, tmp);  // FFT による畳み込みの計算

    for (int n=0; n<N21p; n++)        // 出力用キューへ格納
    {
        IntrGDisable();
        txQueue.Put(tmp[n+N-1]);
        IntrGEnable();
    }
}

#define MK_OLS_Queue
#endif
```

注7：ここで定義したこの二つの関数に相当する関数は CSL の API 関数として提供されているが，その内容は公開されていないため，インライン関数として展開されるかどうかは不明である．したがって，ここではインライン展開されるようにするため，この二つの関数をインライン関数として定義を行った．

▶ 本体のプログラム

本体のプログラム(OverlapSaveQueue.cpp)は，リスト 3(OverlapSaveArray.cpp)に示したものとほとんど同じです．そこで，異なっている箇所を表 3に示します．

▶ OLS_Queue のプログラム

バッファとしてキューを使う方法では，クラス ConvolverFFT の派生クラス OLS_Queue を使います．これをリスト 7(OLS_Queue.cpp)に示します．このクラスでは，入力信号および出力信号のためのバッファをキューで構成します．

非公開部では，入力バッファおよび出力バッファのために使うキューとして，クラス ExQueue のオブジェクト rxQueue，txQueue が宣言されています．メンバ関数としては，グローバル割り込みの許可と禁止を行う関数 IntrGEnable()，IntrGDisable() が定義されています[注7]．この二つの関数は，メンバ関数 Execute() の中で使います．

クラス OLS_Queue のコンストラクタは，メンバ初期設定の機能(“：”以下の部分)を使って基底クラスのコンストラクタへ必要なデータを送り，さらに非公開メンバに値を設定します．それ以外の処理は行っていません．

rxPut() は，入力バッファとして使うキュー rxQueue に入力信号を格納するためのメンバ関数です．

txGet() は出力バッファとして使っているキュー txQueue から出力信号を取り出すためのメンバ関数です．

メンバ関数 Execute() で行われる処理は次のようになっています．

最初に，入力バッファとして使うキュー rxQueue から $M-N+1$(M：FFT の点数，N：フィルタの係数の個数)個のデータをメンバ関数 Get() を使って取り出し，

クラス ConvolverFFT のメンバ関数 Execute() で使う作業領域 tmp の先頭(0番目)から順に格納します．

次に，同じキューから今度はメンバ関数 GetCopy() を使って $N-1$ 個にデータを取り出し，同じ作業領域 tmp に $M-N+1$ 番目から順に格納します．その結果，次にメンバ関数 Get() を使って取り出すときまで，メンバ関数 GetCopy() を使って取り出した $N-1$ 個のデータはまだキューの中に残っていることになります．この残っているデータは，次のブロックの処理で，重複して用いる入力データに相当します．

以上の転送が終了したら，クラス ConvolverFFT のメンバ関数 Execute() で畳み込みの計算を行います．

最後に，クラス畳み込みの結果の中で，先頭の $N-1$ 個を除く $M-N+1$ 個のデータをメンバ関数 Put() を使って出力バッファ用のキュー txQueue へ格納します．

以上の処理で注意しなければならない点は，入力バッファおよび出力バッファとして使用しているキューはメンバ関数 Execute() と受信割り込み処理を行う関数 AIC_RX_ISR() の両者からアクセスされるということです．したがって，メンバ関数 Execute() の中でキューに対する操作を行っている最中に，受信割り込みが発生し，その受信割り込み処理の中でも同じキューに対する処理が行われる可能性があります．このような状態が発生すると，キューに対する操作が正しくできなくなる場合が出てきます．

したがって，ここで使うキューはマルチタスク・システムにおけるクリティカル・セクション(critical section)のように扱わなければなりません．つまり，相互排除ということを考える必要があります．これに対処する方法[注8]はいろいろありますが，ここではもっとも簡単な方法を採用します．つまり，メンバ関数 Execute() の中でキューに対する操作を行っている最中は受信割り込みを禁止するという方法です．そのため，キューに対する操作を行うメンバ関数 Get()，GetCopy()，Put() の前にはメンバ関数 IntrGDisable() を使ってグローバル割り込み[注9]を禁止し，キューに対する操作が終了したらメンバ関数 IntrGEnable() を使ってグローバル割り込みを許可するということを行っています．

● 実行結果

実行結果はどちらのプログラムもまったく同じになります．写真1には0.8kHzの矩形波を入力した場合について示しています．上の波形は入力信号，下の波形は出力信号です．カーソルは矩形波の基本周期を表しており，画面の上方のほぼ中央部にはカーソルの間隔に対応する周波数が表示されています．この表示から矩形波の基本周波数は0.8kHzということがわかります．一方，画面の左上には小さな数値で，下に表示されているフィルタの出力である正弦波の周波数が表示されており，

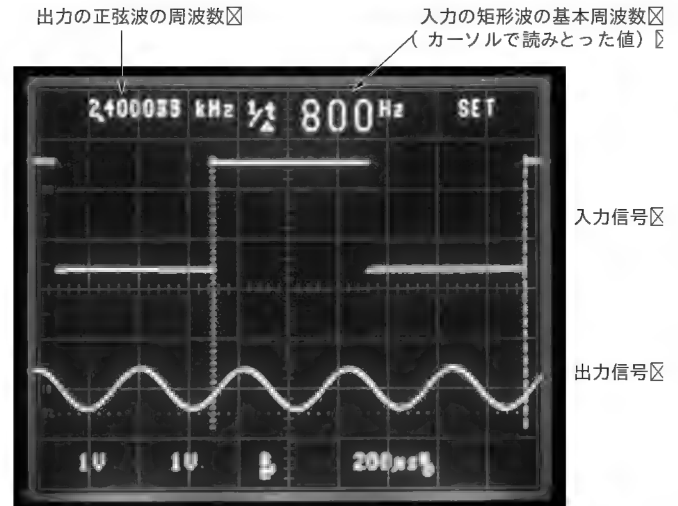


写真1 FFTによるFIRフィルタの実行結果

2.4kHzであることがわかります．

この結果は次のように解釈できます．矩形波は基本周波数とその奇数倍の成分を持ちます．したがって，0.8kHzの矩形波は0.8，2.4，4.0，5.6，…kHzという周波数成分を持つことになります．ここで作成したFIRフィルタは，この周波数成分の中で2.4kHzの成分のみを通します．したがって，フィルタの出力には写真1の下に表示されている波形のように2.4kHzの正弦波が得られることになります．

＊　　＊

次回は，FFTクラスの応用としてリアルタイム・スペクトル・アナライザのプログラムを作成します．

参考文献

(1) 三上 直樹："C言語によるディジタル信号処理入門"，CQ出版社，2002年．

(2) 瀬谷 啓介："DSP Cプログラミング入門"，技術評論社，2000年．

(3) *TMS320C6000 Chip Support Library API User's Guide*, Texas Instruments, 文献番号：SPRU401F，2003年2月．

みかみ・なおき　職業能力開発総合大学校 情報工学科

注8：実際のリアルタイムOSでは，セマフォ(semaphore)やミューテックス(mutex)を使う方法がよく利用される．

注9：このプログラムの場合は受信割り込みだけを禁止すればよいので，割り込みマスクの処理だけでよく，グローバル割り込みを禁止する必要はない．しかし，グローバル割り込みを禁止するほうが，プログラムが多少は簡単になるので，ここではグローバル割り込みを禁止するようにした．

VBと親和性が高く，機械制御を容易に実行できる PCベースのマルチプロセス・コントローラの開発

古川 昌志/片桐 崇

1 はじめに

産業用機械装置のユーザ・インターフェースやその使い勝手，ネットワーク化などに対する要求は年々高度化しています．従来から，産業用機械装置のコントローラとして，PLC(Programmable Logic Controller)が使用されていますが，最近ではWindows PCを使用することも多くなっています．

PLCに対してPCを使用するメリットは，

- ▶安価で高性能なハードウェアが市販されていること
- ▶Visual C++(以下，VC++と記す)などの高級言語を使用することでソフトウェアの資産化を図れること
- ▶Visual Basic(以下，VBと記す)などのRAD(Rapid Application Development)ツールを使用することで高度なユーザ・インターフェースを比較的容易に開発できること

などがあります．

しかし，VC++を使用した開発は，一般にMFC(Microsoft Foundation Class)の敷居が高いといわれており，"だれでも簡単に"というわけにはいかないのが現状のようです．一方，VBを使用した開発では，マルチスレッドを使用することが難しいため，複雑な機械制御を行うことは困難です．

そこで，筆者らは，VBと親和性が高く，機械制御を容易に実現できるマルチプロセス・コントローラを開発しました．以下，本コントローラをMWC(MOTIWARE Controller)と記します．MWCはWindowsのアウト・プロセスCOMサーバとして動作します．

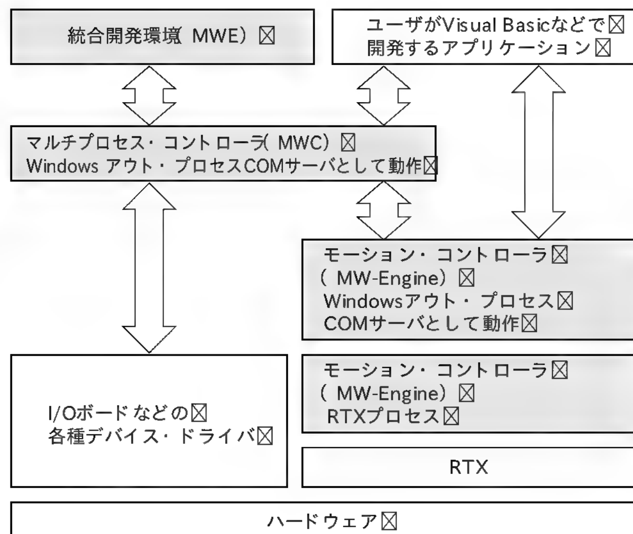


図1 マルチプロセス・コントローラMWC(MOTIWARE Controller)のシステム構成

2 マルチプロセス・コントローラのシステム構成

図1はMWCのシステム構成です．

MWCはRTXという，Windowsでリアルタイム制御を可能にするための拡張機能を搭載しており，RTXのプロセスとして，モーション・コントロール機能があります〔以下，MW-Engine(MOTIWARE Engine)と記す〕．

MW-EngineのRTXプロセスと，MW-EngineのCOMサーバ間は，RTXが提供するプロセス間通信機能でインターフェースされます．MW-EngineのCOMサーバの上位には，独自に開発したC言語準拠のプログラム〔以下，MOS(MOTIWARE Script)言語と記す〕をマルチプロセスで実行可能なマルチプロセス・コントローラCOMサーバがあります．さらに，MWCでのアプリケーション開発を効率よく進めるため，統合開発環境〔以下，MWE(MOTIWARE Environment)と記す〕が用意されています．

3 RTX――Windows(NT/2000/XP)をRTOSとして使用可能とするツール

RTXは米Venturcom社が開発した，Windowsをリアルタイム拡張するためのツールです．PC向けのWindowsにRTXをインストールすることで，リアルタイム性が要求されるアプリケーションにWindows搭載PCが使用できるようになります．

RTXを使用したアプリケーションは，RTXプロセスとWin32プロセスで構成されます．RTXプロセスとWin32プロセスはプロセス間通信でインターフェースされます．RTXプロセスは，RTXの実行環境で動作し，ハード・リアルタイム性が要

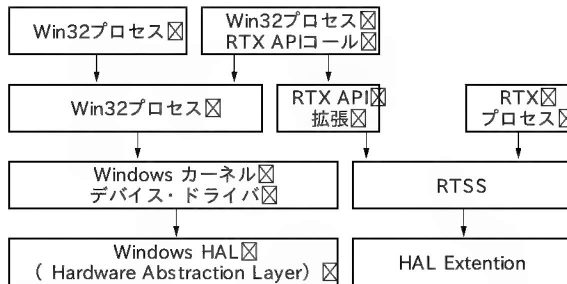


図2 RTXのアーキテクチャ

求される部分を担当します．Win32プロセスは，RTXプロセスに処理要求を発行したり，RTXプロセスで発生したデータを表示するために使用します．

図2にRTXのアーキテクチャを示します．

RTXプロセスは，VC++6.0を使って開発可能で，.RTSS(Realtime Subsystem)ファイルとしてビルドされます．デバッグ・ビルドしたEXEファイル，およびRTSSファイルはVC++6.0でのソース・デバッグが可能ですが，Win32環境でのデバッグになるため，リアルタイム性がありません．そのため，プロセス，スレッドの実行順序はリアルタイム環境と異なることに注意が必要です．RTSSファイルをリアルタイム環境でソース・デバッグする場合はWinDBGを使用します．

RTXは，Win32APIと同じ，あるいは類似のAPIが用意されており，習得は容易です．また，使い慣れたVC++を使って開発できることも，開発期間の短縮に役立ちます．

4 モーション・コントロールのしくみ

MWCでは，RTXを使用したモーション・コントロール機能(MW-Engine)を搭載しています．MW-Engineは，独自に開発したPCIボードの1msタイマ割り込み処理で各メカニズムの目標位置パルスを求めるため，一般的なパルス列発生ボードでは実現不可能な複雑な機構をもったメカニズムの軌跡制御が可能となっています．

● ソフトウェアの構成

MW-Engineは，複数のメカニズムを同時に8個まで独立して制御することができます．スレッド構成は，

- ▶ 1msタイマ割り込みで実行されるモーション制御の実時間処理スレッド
- ▶ 各メカニズムに対する同期的な処理要求を実行するスレッド
- ▶ 全メカニズムに共通した，非同期的な処理要求を実行するスレッド

から構成されます．

Win32プロセスで共有メモリにパラメータとコマンド・コードを書き込んだあと，イベントを発生させ，コマンド要求をRTXのメカニズムn用タスクに通知します．RTXのメカニズムn用タスクは，要求された処理を実行後，応答データを共有メモリに格納し，応答イベントを発生させ，コマンド終了をWin32プロセスに通知します(図3)．

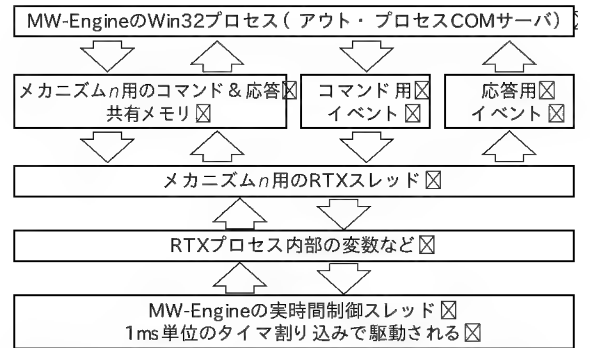


図3 MW-Engine(MOTIWARE Engine)のアーキテクチャ

● コマンドと機能

表1にMW-Engineが提供するインターフェースとメソッドのおもなものを示します．MWCのモーションに関するシステム関数は，MW-Engineの提供するインターフェースのメソッドを呼び出すことで実行されます．MW-Engineのインターフェースは VC++から直接使用することもできるため，VC++で高度なモーション制御アプリケーションを開発することも可能となっています．

● 設定ファイル

MW-Engineでは装置の物理的構成，たとえば，モータ・パルス数と減速比の関係，モータの回転方向と軸の進行方向の関係などを設定ファイルにパラメータ化し，MW-Engineを利用する場合に，これらの物理的な事項を意識しなくてもよいようにしています．

これにより，ハードウェアや機械構成の変更に対してMW-Engineのアプリケーションを堅牢にすることができます．設定ファイルの代表的な項目を表2に示します．

5 C言語準拠トランスレータとインタプリタ

● トランスレータの役割

MOS言語のトランスレータ(以下，MOSトランスレータと記す)は，基本的にC言語に準拠していますが，構造体をサポートしていないなどの制限があります．また，C言語のようなヘッダ・ファイルはありません．MOSトランスレータは，まず，ソース・プログラムの字句解析を行い，トークン・リストを作成します．次に，作成したトークン・リストをたどり，構文解析と中間コードの生成を行います．

MOSトランスレータは，前方参照を行うことができないため，ソース・プログラムで使用する変数，定数，ユーザ関数は，使用する前(プログラムの行数が小さいほう)に，あらかじめ定義されている必要があります．

MOSトランスレータは機械語のプログラムを生成するコン

表1　主要なMW-Engineのインターフェース

インターフェース	メソッド	機　能
ISystemCtrl	EnableErrorException	エラー時の例外スローの有効/無効
	GetAxesEnable	軸の有効/無効の取得
	GetFatalErrorStatus	フェイタル・エラー状態の取得
	GetRobotEnable	メカニズムの有効/無効の取得
	GetRobotErrorStatus	メカニズム・エラー状態の取得
	GetSystemErrorStatus	システム・エラー状態の取得
IRobotCtrl	ArcMove	円弧補間動作
	CircularMove	真円補間動作
	GetAxesEnable	軸の有効/無効の取得
	GetCarteGoal	XYZ座標系の最終目標位置の取得
	GetCartePosition	XYZ座標系の現在位置の取得
	GetHomeSensorStatus	原点センサ状態の取得
	GetJointGoal	軸座標系の最終目標位置の取得
	GetJointPosition	軸座標系の現在位置の取得
	GetRobotErrorStatus	メカニズム・エラー状態の取得
	GetServoPowerStatus	サーボ電源状態の取得
	GetTravelLimitSensorStatus	オーバラン・センサ状態の取得
	JogMove	XYZ座標系で任意方向へJOG動作
	JogJointMove	軸座標系で任意方向へJOG動作
	PtpJointMove	軸座標系でPTP動作
	PtpMove	XYZ座標系でPTP動作
	ReturnHome	原点復帰動作
	SetCpAccelerations	補間動作の加減速パラメータ設定
	SetCpSpeed	補間動作の速度設定
	SetPtpAccelerations	PTP動作の加減速パラメータ設定
	SetPtpSpeed	PTP動作の速度設定
	SetRobotNo	制御対象メカニズムの指定
	SetServoPower	サーボ電源のON/OFF

表2　MW-Engineの設定ファイルで設定可能なパラメータ（おもなもの）

システム全体	実機動作モード/シミュレータ・モード
各メカニズム	構成軸数
	メカニズムの機構
	アーム長さ
各軸	エンコーダの有無，エンコーダ・パルス数
	軸の1移動単位あたりのパルス数
	最高パルス列出力周波数
	原点復帰方法（原点センサ使用，OTセンサ使用，など）
	原点センサの有無，論理
	オーバ・トラベル（OT）センサの有無，論理

パイラではありません．中間コードを生成し，インタプリタが中間コードを解釈して実行します．

● パラメータ・スタックとコール・スタックをもつインタプリタ

MOSインタプリタは，内部的に2種類のスタックを使用します．パラメータや，関数の返却値を格納するために使用するのがパラメータ・スタックです．ユーザが定義した関数からの戻り番地を格納するために使用するのがコール・スタックです．

1）パラメータ・スタック

MOSインタプリタは，中間コードを実行した結果をパラメータ・スタックに格納します．`val = a+b;` が実行されるとき，中間コードは下記のように逆ポーランド表現で生成されます．かっこ内は，中間コードを実行したあとのパラメータ・スタックの段数です．

- 変数valのアドレス・ロード（1）
- 変数aの値ロード（2）
- 変数bの値ロード（3）
- 演算子+（2）
- パラメータ・スタックの1stをパラメータ・スタックの2ndに格納（1，計算結果がパラメータ・スタックに残る）
- 文末記号 ';'（0，パラメータ・スタックを1段捨てる）

2）コール・スタック

MOSインタプリタは，ユーザ関数をコールする場合，戻り位置を保存するため，コール・スタックを使用します．また，コール・スタックは，ユーザ関数に引き渡される引き数と，ユーザ関数のローカル変数の領域を管理し，ユーザ関数からリターンするときにこれらを破棄します．

● 例外処理時に必要なランタイム・エラー

機械装置の制御を行う場合，気を付けなければならないのが，非常停止などの例外処理の頻度が多いという点です．

MOSインタプリタでは，非常停止状態でメカニズム動作関数を使用した場合などには，ランタイム・エラーとなり，MOSインタプリタの実行が止まります．この機能により，MOSプログラムは，不要なエラー処理から解放されます．

6 マルチプロセス・コントローラの機能

● マルチプロセスとは

VBで機械装置のソフトウェアを開発する場合の最大のネックは，マルチスレッドが容易に使用できないことです．MWCでは，MOS言語で記述したプログラムを同時に10個まで実行することができます．

各MOSプロセスは，後述する共有資源を介してインターフェースされますが，各MOSプロセスの変数を直接参照することはできません．その意味で，マルチスレッドではなく，マルチプロセスということになります．

複数のMOSプロセスを同時に実行することで，機械的な結

合度が疎である要素を，個別のプロセスで実行することができます．たとえば，装置の搬入側，搬出側にコンベアがある場合など，各コンベアを個別のMOSプロセスで実行することで，メインの装置とは切り離したプログラムとなり，プログラムが見やすくなり，メンテナンス性も向上します．

● 共有資源

MWCは，各MOSプロセスおよびVBプロセスで共有可能な資源を提供します．

1) 共有変数

MOSプロセス間，または，MOSプロセスとVBプロセス間で変数を共有するための機能です．MOSプロセス，またはVBプロセスからの共有変数へのアクセスは互いに排他されます．したがって，各プロセスでは，複数プロセスの処理であることを特に意識することなく，共有変数への読み書きを行うことができます．

2) 物理I/O

入力が1～512まで，出力が513～1024までの，各512点が用意されています．実際にPCに実装されている入出力ボードに合わせてDLLを登録することで，任意の入出力ボードへの対応が可能となっています．

3) 論理I/O

1025～2048までの1024点が論理I/Oとして用意されています．MOSプロセス間，またはMOSプロセスとVBプロセス間の状態フラグとして使用します．

● 用意されているシステム関数

MOS言語に用意されているシステム関数は，数学関数，角度変換，モーション，入出力，タイマ，共有変数，イベントに分類できます(表3)．

● COMインターフェース

MWCはCOMインターフェースを公開しています．VBプロセスは，このインターフェースのメソッドなどを使用して，MWCのMOSプロセスを起動停止したり，共有資源へのアクセスを行います(表4)．

表3　MOS言語のおもなシステム関数

数学関数		Abs，acos，asin，atan，atan2，ceil，cos，log，fabs，floor，Log10，pow，sin，sqrt，tan
角度変換		RadToDeg，DegToRag，pi
モーション	動作制御	RobPtpMove，RobReturnHome，RobStopMove，RobSetServoPoert，RobLinearMove，RobArcMove
	設定	RobSetPtpSpeed，RobSetPtpAccelerations RobSetSettle，RobSetCpSpeed
	座標	RobGetCarteGoal，RobGetCartePosition
	ステータス	RobWaitForMoveDone RobGetMoveStatus
I/O		ReadPort，WritePort
タイマ		StartTimer，GetTimerValue，Sleep
共有変数		ReadSharedVariable，WriteSharedVariable
標準出力		Printf0，Printf1，Prinf2
イベント		FireEvent

表4　MWC(MOTIWARE Controller)のインターフェース

インターフェース	メソッド	機　能
ISystemCtrl	Start	MOSプロセスの初期起動
	Stop	MOSプロセスの停止
	QueryExecLineInfo	MOSプロセスの現在実行行の取得
	QueryProcStatus	MOSプロセスの状態取得
	WritePorts	入出力ポートへの状態設定
	ReadPorts	入出力ポートからの状態取得
	WriteVariable	MOSプログラムの変数値への書き込み
	ReadVariable	MOSプログラムの変数値の読み出し
	WriteSharedVariable	共有変数への書き込み
	ReadSharedVariable	共有変数からの読み出し
	QueryProcRuntime Error	MOSプロセスのエラー情報取得
	ApplEvent	MOSプログラムのFire Eventに対応

7　効率よく開発するための統合開発環境

● 統合開発環境のシステム概要

MOSプログラムを効率よく開発するために，統合開発環境(MWE)を開発しました．図4に統合開発環境MWEの実行画面を示します．

統合開発環境MWEは，MWCの提供するCOMインターフェースを使用して作成されています．ユーザが作成するVBプロセスでも，まったく同様のCOMインターフェースを使用できるようになっています．

以下におもな機能と役割を示します．

1) ソース・エディタ
- MOSプログラムの編集，実行をトレース

2) プロセス管理
- プロセスの割り付け

3) プロセス制御(図5)
- 各プロセスの起動，停止，オブジェクトのロード
- 変数のリード/ライト
- MOS言語のビルド

4) 共有変数
- 共有変数のリード/ライト

5) 入出力モニタ(図6)
- ディジタル入出力，論理I/Oの状態取得と状態設定

6) 標準出力
- printf0などで出力したメッセージを表示

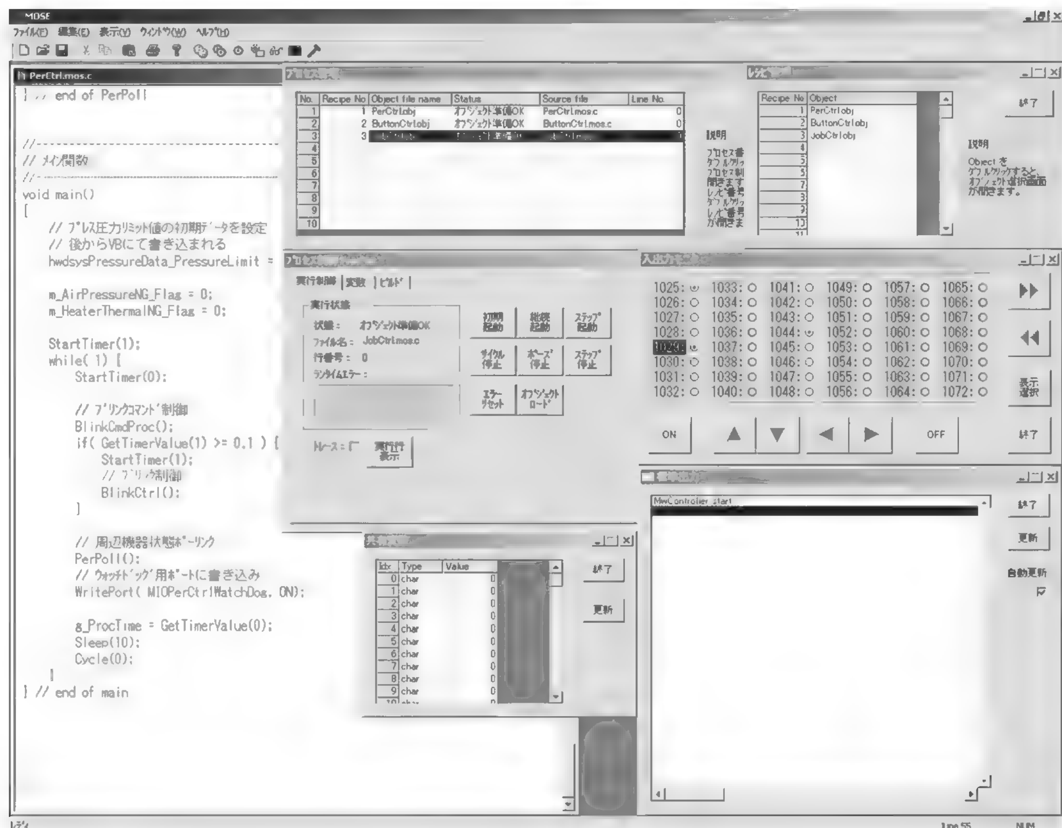


図4　統合開発環境 MWE の実行画面

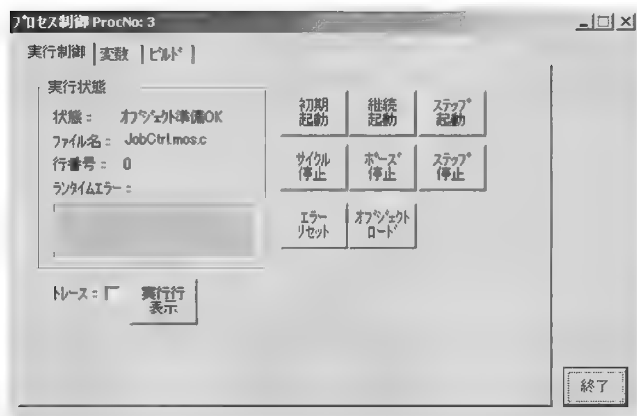


図5　プロセス制御

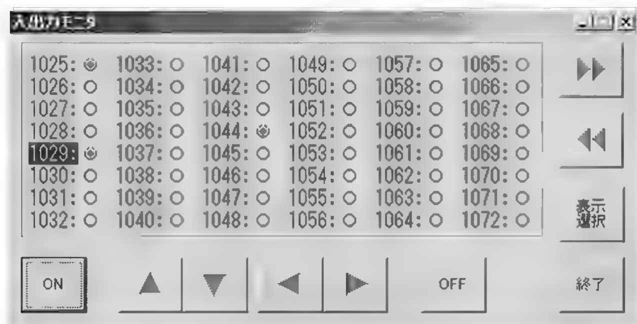


図6　入出力モニタ

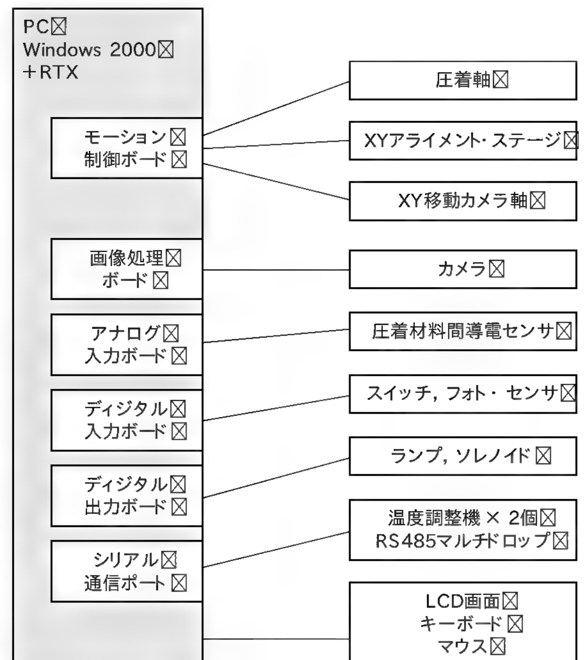


図7　装置の構成

8 機械制御アプリケーションへの応用

今回作成した統合開発環境をロボットにワーク(作業対象)を移動カメラで検出させ，それをプレス圧着させるという機械制御に応用した例を紹介します．

● 制御する装置の構成

統合開発環境MWCを図7のような構成の装置に適用します．

ターゲットの装置は，XY移動カメラ，XYアライメント・ステージ，圧着軸からなります．そのほかに，圧着材料間導電センサとアナログ入力ボード，ワーク(作業対象)の温度を制御するための温度調節機がシリアル通信ポートに接続されています．装置を操作するための各種ボタンとそのランプ，各アクチュエータを駆動するためのソレノイドとフォト・センサが，ディジタル入出力ボードに接続されています(表5)．

● ソフトウェア構成

装置の制御用ソフトウェアの構成を図8に示します．

このソフトウェアはVBプロセス，MOSプロセス1，MOSプロセス2，MOSプロセス3から構成されています．

表5 圧着部仕様

項　目	仕　様
最大ワーク・サイズ	φ 50mm 高さ 50mm
最大加圧力	30kN
センサ・フィードバックによる圧着軸制御周期	1ms
温度制御	上ワーク，下ワーク
ワーク自動アライメント	下ワーク

1) VBプロセス
- 画面に各種の情報を表示
- レシピ編集，装置パラメータ編集
- 手動操作
- 画像処理

2) MOSプロセス1
- ボタン/ランプの点滅
- 非常停止，異常などのエラー監視

3) MOSプロセス2
- ボタンが押されたことを検出し，VBプロセスに対してイベントを発行

4) MOSプロセス3

VBプロセスから要求された各種の処理
- 装置原点復帰
- 自動アライメント処理
- 圧着軸
- その他

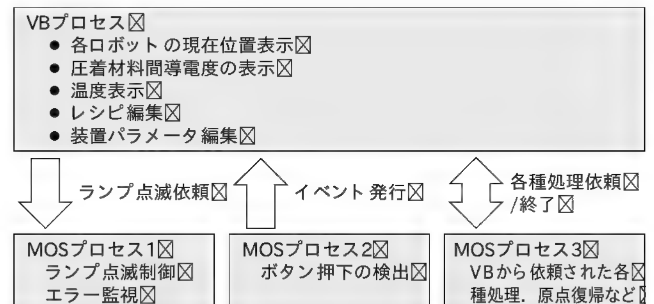


図8 装置の制御用ソフトウェアの構成

リスト1 ボタン押下の検出

```
const short MaxButtonCtrlBlockItem = 100;
int     g_ButtonCtrlBlock[ MaxButtonCtrlBlockItem ][5];
double  g_ProcTime;


int ScanButtons()
{
    int rc;
    int i;
    int port;

    rc=0;
    i=0;
    while( (i < MaxButtonCtrlBlockItem ) && (rc == 0) ) {
        if( g_ButtonCtrlBlock[i][0] != 0 ) {
            // ポート番号
            port = g_ButtonCtrlBlock[i][0];
            // ポート番号の入力状態が指定値と合致するか調べる
            if( ReadPort( port) == g_ButtonCtrlBlock[i][1] ) {
                // チャタリング防止のため，3回連続して
                // 所望の状態であるか調べる
                g_ButtonCtrlBlock[i][2] =
                                   g_ButtonCtrlBlock[i][2]  + 1;
                if( g_ButtonCtrlBlock[i][2] == 3 ) {
                    // ボタンが押された旨，リターン値として設定する
                    rc = port;
                }
            }
            else {
                g_ButtonCtrlBlock[i][2] = 0;
            }
        }
        i = i+1;
    }

    return( rc);
} // end of ScanButtons


void main()
{
    int rc;

    while( 1) {
        StartTimer(0);
        // ボタンが押されたか否かを検出する
        rc = ScanButtons();
        if( rc != 0 ) {
            // VBに対して，押されたボタンに対応したイベントを発行する
            FireEvent( rc);
        }
        g_ProcTime = GetTimerValue(0);
        Sleep(20);
        Cycle(0);
    } // while

} // end of main
```

リスト 2　イベント処理

```
' JOBコマンド実行
Private Sub JobCmdExecute(CmdCode As Integer)

    ' 画面無効にする
    Me.Enabled = False
    ' 共有変数に VB から MOS への処理依頼コードを書き込む
    Call objMwcSystemCtrl.WriteSharedVariable(
                    JobCtrlCmdSharedVariable, CVar(CmdCode))
End Sub


' ボタン押下のイベント
Private Function ButtonEventProc(EventID As Long) As Integer
    Dim rc As Integer

    rc = 0

    Select Case EventID
    Case PerCtrlPI_ReturnHome_Button
        ' 原点復帰ボタン
        JobCmdExecute (JobCtrlCmd_ReturnHome)
        rc = 0

    Case ...
        ...
    case ...
        ...

    End Select

    ButtonEventProc = rc
End Function


' MOS から VB へのイベント
Private Function objMwcSystemCtrl_ApplEvent(
                        _ByVal ProcNo As Integer,
                        _ByVal EventID As Long) As Integer

    objMwcSystemCtrl_ApplEvent = 0

    Select Case ProcNo
    Case MoseProc_ButtonCtrl ' MOS プロセス 2 からのイベント
        ' ボタン押下のイベント
        ' 各ボタンに対応した処理を行う
        objMwcSystemCtrl_ApplEvent = ButtonEventProc(EventID)
    Case MoseProc_JobCtrl    ' MOS プロセス 3 からのイベント
        ' Job の結果イベント
        objMwcSystemCtrl_ApplEvent = JobResEventProc(EventID)
    End Select

End Function
```

リスト 4　VB から MOS への変数値の受け渡し

```
' 指定インデックスのレシピ・データを MOSE に送る
Public Function SendRecipeForMose(idx As Integer) As Integer
    Dim rc As Integer
    Dim proc As Integer

    On Error GoTo Error

    ' レシピを送る対象の MOSE プロセス番号
    proc = MoseProc_JobCtrl

    '============================================
    ' カメラ位置データ
    '============================================
    ' P1:上:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos1UX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos1UX))
    ' P1:上:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos1UY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos1UY))

    ' P2:上:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos2UX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos2UX))
    ' P2:上:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos2UY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos2UY))

    ' P3:上:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos3UX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos3UX))
    ' P3:上:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos3UY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos3UY))

    ' P1:下:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos1LX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos1LX))
    ' P1:下:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos1LY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos1LY))

    ' P2:下:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos2LX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos2LX))
    ' P2:下:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos2LY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos2LY))

    ' P3:下:X
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos3LX", CVar(m_Recipe(idx).m_Camera_Alignment_Pos3LX))
    ' P3:下:Y
    rc = objMwcSystemCtrl.WriteVariable(proc, "recipeCamera_Alignment_Pos3LY", CVar(m_Recipe(idx).m_Camera_Alignment_Pos3LY))

    ...

    SendRecipeForMose = 0
    Exit Function

Error:
    SendRecipeForMose = 1

End Function
```

リスト 3　VB からの処理要求処理

```
//------------------------------------------
// VB からの処理要求の取得
//------------------------------------------
int GetJobCtrlCmd()
{
    // 共有変数の値をリードする
    return( ReadSharedVariable( JobCtrlCmdSharedVariable) );
}


//------------------------------------------
// VB からの処理要求に対する成功終了応答
//------------------------------------------
void SetJobCmdReplySuccess()
{
    // 共有変数に値0 を書き込む
    WriteSharedVariable( JobCtrlCmdSharedVariable
                                              + 1, 0/*Success*/);
}

//------------------------------------------
// VB からの処理要求に対する失敗終了応答
//------------------------------------------
void SetJobCmdReplyFail( int ErrCode)
{
    // 共有変数にエラー値を書き込む
    WriteSharedVariable( JobCtrlCmdSharedVariable + 1, ErrCode);
}

//------------------------------------------
// Jobコマンドの終了イベント発行
//------------------------------------------
void FireJobCmdReplyEvent( int CmdCode)
{
    // コマンド終了
    WriteSharedVariable( JobCtrlCmdSharedVariable, 0);
    // イベント発行
    FireEvent( CmdCode);
}


//------------------------------------------
// Jobコマンド: 原点復帰
//------------------------------------------
void CmdProc_ReturnHome()
{
    double pos[MaxRobAxes];

    if( CheckRobServo() != 0 ) {
        // ロボット・サーボ OFF
        Printf0( "原点復帰: ロボットサーボ OFF¥n");
        // コマンド失敗
        SetJobCmdReplyFail( JobCtrlErr_RobotServoOff);
        // コマンド終了通知イベント
        FireJobCmdReplyEvent( JobCtrlCmd_ReturnHome2);
        return;
    }

    // 装置原点復帰未完了にする
    SetReturnHomeSts( 0);

    // 原点復帰ランプをブリンク開始
    StartBlink( PerCtrlPO_ReturnHome_Lamp, 1/*100msec*/);

    // 周辺装置に関する処理など
    ...
    ...
    // 圧着軸: 原点復帰
    RobReturnHome( RobPress);
    // カメラ: 原点復帰
    RobReturnHome( RobCamera);
    // ステージ: 原点復帰
    RobReturnHome( RobStage);

    // 周辺装置に関する処理など
    ...
    ...

    // 原点復帰ランプをブリンク終了
    EndBlink( PerCtrlPO_ReturnHome_Lamp);
    // 原点復帰ランプを点灯
    WritePort( PerCtrlPO_ReturnHome_Lamp, ON);
    Printf0( "原点復帰:正常終了¥n");

    // 装置原点復帰完了
    SetReturnHomeSts( 1);

    // コマンド成功
    SetJobCmdReplySuccess();

    // コマンド終了通知イベント
    FireJobCmdReplyEvent( JobCtrlCmd_ReturnHome);

} // end of CmdProc_ReturnHome


//------------------------------------------
// Jobコマンド実行メイン処理
//------------------------------------------
void main()
{
    int cmd;

    // プログラムの初期化
    Initialize();

    while( 1) {
        // VB からの処理要求を取得
        cmd = GetJobCtrlCmd();
        switch( cmd) {
        case 0:                          // コマンドなし
            Sleep(20);
            break;
        case JobCtrlCmd_ReturnHome:      // 原点復帰
            CmdProc_ReturnHome();
            break;

        case ...
            ...
        case ...
            ...

        default:                         // 未定義コマンド
            Printf0( "未定義コマンド¥n");
            // コマンド失敗
            SetJobCmdReplyFail( JobCtrlErr_UndefinedCmd);
            // コマンド終了通知イベント
            FireJobCmdReplyEvent( cmd );
            break;
        } // switch

        Cycle(0);
    } // while

} // end of main
```

●ソフトウェアの処理

ボタンが押されたことをMOSプロセス2が検出すると，そのことをMOSシステム関数 FireEvent にてVBに通知します(リスト1，p.177)．

MOSプロセス2で FireEvent を実行したとき，VBでは ApplEvent というイベントが発生します．ApplEvent では，引き数に FireEvent 関数を実行したMOSプロセス番号と FireEvent 関数の引き数として引き渡した値が得られます(リスト2)．

VBプロセスでは，得られたMOSプロセス番号と，引き数値によりボタンが押されたことによる各種の処理を行います．たとえば原点復帰ボタンが押された場合，VBプロセスはMOSプ

リスト 5　MOSプログラムでのレシピ・データの定義

```
//==============================================
// カメラ位置データ
//==============================================
// P1:上:X
double recipeCamera_Alignment_Pos1UX;
// P1:上:Y
double recipeCamera_Alignment_Pos1UY;

// P2:上:X
double recipeCamera_Alignment_Pos2UX;
// P2:上:Y
double recipeCamera_Alignment_Pos2UY;

// P3:上:X
double recipeCamera_Alignment_Pos3UX;
// P3:上:Y
double recipeCamera_Alignment_Pos3UY;


// P1:下:X
double recipeCamera_Alignment_Pos1LX;
// P1:下:Y
double recipeCamera_Alignment_Pos1LY;

// P2:下:X
double recipeCamera_Alignment_Pos2LX;
// P2:下:Y
double recipeCamera_Alignment_Pos2LY;

// P3:下:X
double recipeCamera_Alignment_Pos3LX;
// P3:下:Y
double recipeCamera_Alignment_Pos3LY;

...
```

ロセス3に対して，原点復帰実行の処理要求を発行します．

処理要求は，MWCの共有変数機能を使用して行われます．MOSプロセス3では，VBプロセスからの原点復帰要求を受けて，原点復帰ボタンの点滅を開始し，圧着軸，移動カメラ，アライメント・ステージの順番で原点復帰を行います．周辺装置の原点復帰を行い，原点復帰ボタンのランプを点灯後，VBプロセスからの処理要求が終了したことをイベントにて通知します(リスト3，p.179)．

VBプロセスでは，処理要求の終了イベントを受けて，必要なメッセージを画面に表示します．

ハードウェア・ボタンが押された場合について説明しましたが，VBプロセスの画面内に設けた各種の操作ボタンを押した場合も，同様にVBプロセスから，MOSプロセス3に対して処理要求を発行します．

● レシピ・データの共有

品種ごとに異なった値であるパラメータは，レシピ・データにより管理を行います．VBプロセスではレシピの編集と，保存，読み出しを行います．

VBプロセスで編集したレシピ・データは，MWCの変数リード/ライト機能により，VBプロセスとMOSプロセス間で，レシピ・データの授受を行います．

リスト4(p.178)では，VBプロセスから，MOSプロセスの変数に値を書き込む処理を行っています．

リスト5は，MOSプログラムでのレシピ・データの定義部です．

● 装置パラメータの共有

レシピ・データのように，品種ごとに異なった値ではありませんが，必要に応じて変更が必要なパラメータは，装置パラメータとして管理を行います．VBプロセスでは，装置パラメータの編集と，保存，読み出しを行います．VBプロセスで編集した装置パラメータは，レシピ・データと同様の方法で，MOSプロセスに書き込みます．

おわりに

以上，機械装置のソフトウェア開発に関する新たな一手法について説明してきました．本稿が，機械制御のソフトウェア開発のヒントとなれば幸いです．

参考文献

(1) 佐藤隆浩，Windows NT/Windows 2000/Windows XPをRTOSとして使用可能にする　Windowsのリアルタイム拡張RTX，Interface，2001年12月号

ふるかわ・まさし　(有)タンデム
かたぎり・たかし　(有)プライムモーション
問い合わせ先：support@primemotion.com

IPパケットの隙間から

Copyrightについて考えてみよう

祐安 重夫

DVDを買いに行ったついでにCDを買っていこうかと思ったら，あきれたことに最近発売されたCDの形をしたものは，ことごとくCCCD（Copy Control CD）ではないか．CDを買いにきたのであって，まがい物を買いに来たのではない．これを中国や日本では，昔から「羊頭狗肉」ということばで表現する．結局，買わずに帰った．

- CCCDは音楽CDの標準規格に準拠していないので音楽CDではない．
- CCCDはその仕様が公開されていないので，CCCD用のプレーヤも存在しない．CDプレーヤのメーカはCCCDの再生を一切保証していない．
- CCCDは意図的に過度のエラーを混入させることで，CDプレーヤのエラー補正機能を利用して再生させることを期待しているので，CCCDを既存のプレーヤで再生できるという保証はまったくないし，場合によってはCDプレーヤに過剰な負荷をかけることがある．プレーヤの構造によっては，ハードウェアにまで負荷をかけ，寿命を短くしたりハードウェアを破損することもありうる．
- CCCDを販売しているレコード・メーカは，再生できないという理由での引き取りや交換を，一切拒否している．

つまり，まがい物の不良品を，消費者に押しつけているのがCCCDであり，さらに意図的にエラーを混入させるという方式の結果として，仮に再生できたとしても低音質のものをわざと販売しているということである．

では，なぜこのようなCCCDなどというものが販売されるようになったのかというと，インターネット上でのファイル交換が盛んになって，特にP2Pに対応したソフトウェアによってそれが助長されていることを，音楽産業側は理由としてあげている．つまり「著作権者」である音楽産業の利益を保護するというのが，その理由である．

だが，待ってほしい．音楽を作ったのは「著作者」である音楽家である．このことが必ずしも「著作者」の利益を守っているわけではない．たとえば（あくまでも仮にだが），世間知らずの若い音楽家を丸め込んで，「著作者」に不利な形で「著作権」のほとんどの部分を手に入れた音楽産業があったとして，音楽産業の利益は本当に「著作者」の利益になっているのだろうか．中には「上演権」まで音楽産業に取られてしまって，自分たちの作った曲を自由に演奏できない「著作者」がいるという話さえ聞く．

GNUプロジェクトのことを考えてみよう．以前にもどこかで書いたことがあるのだが，GPL，LGPL，Copyleftといった概念が意味するものは，著作物を著作者の意図するとおりに流通させるということであり，その意味ではこれほど著作権の本来の意義を反映させているものはないのである．

音楽家にとって，それでお金が入ってくることは重要なことだが，同時に自分の曲を多くの人に聞いてもらうことも同じように重要である．それは元作曲家であった筆者自身の意見でもある．その意味では，P2Pを全面的に否定する理由は，必ずしもないのである．

そもそも，P2Pの技術は，著作権法やその他の法律にとって違法なファイルを交換することが本来の目的ではない．特定のサーバを利用することなくクライアントどうしを直結することで，ネットワークへの負荷を分散するという，インターネットの今後のありようにとって重要なものである．IP電話も同様の技術の産物である．

ところで，P2Pといえば有名なのは，Winnyである．Linuxでは動作しないので，筆者は使ったことがないし，これまであまり縁のないソフトウェアであった．

Winnyのユーザが著作権法違反で逮捕されたという事件は，昨年の暮れ近くから聞いてはいたが，まさかWinnyの開発者が著作権法違反の幇助で逮捕されることになるとは，まったく想像もしていなかった．逮捕された金子勇氏は，IPAの「未踏ソフトウェア創造事業」にも参加していた，優秀なプログラマである（もちろん，このプロジェクトでWinnyを開発していたわけではないが）．

ところで，この逮捕を指揮したとされる京都府警生活安全企画課長の阿波拓洋という人物であるが，公安畑出身の警察官僚だという（この逮捕劇のプロセス自体が，いかにも公安的な陰険さに満ちているように感じられる）．

その後，逮捕理由として，金子氏が「現行のディジタル・コンテンツのビジネス・スタイルに疑問を感じていた．警察に著作権法違反を取り締まらせて現体制を維持させているのはおかしい．体制を崩壊させるには，著作権侵害を蔓延させるしかない」と供述したとほざいているが，内容はさておき，表現はいかにも警察が作文しそうなものである．

すでに1960年代後半から，筆者を含む多くの人々がメディア産業の興隆に危機感を持ち，それに対する危惧を表明してきた．その意味では，インターネットの普及とP2P技術の出現で，やっとそれが単なる異義申したてから実効力を持った行動の可能性へと結び付いてきたのである．このコラムでは，金子氏の逮捕が不当逮捕であるという立場から，この問題に注目していきたい．

すけやす・しげお　インターメディア・アクセス

シニアエンジニアの技術草子 四拾壱之段

◆似て非なるもの

旭 征佑

● 遺伝子の神秘

人は自分に似た相手を好む傾向があるという．話題や趣味，洋服のセンスなどが似通った人間に親しみを感じるのは，当然のことだろう．それだけではない．本人は気が付かないが，実はわずかに顔が似ている，そんなことだけでも強い親近感を感じるという．これは，一目惚れの法則などと呼ばれるらしい．犬も飼い主に似てくるといわれるが，実は飼い主が自然と自分に似た犬を買い求めているのだというのは，よく聞く話だ．このあたりは心理学的にも確認されている事実で，「類似性理論」と呼ばれるらしい．

逆に自分と正反対の人間に対しては，興味はもつが，最初は近付き難いところがある．ところが，自分に足らないところを他人で補完するというのは，生活力を高めるためにもっとも重要な条件なのだ．たとえば，趣味が同じということで付き合い始めた2人でも，男性は背が高く腕力が強いため高いところの荷物が動かせる，女性は細かいところに気が利いて料理もうまい…といったように，お互い補完できる機能を無意識に認め合うことで，さらに親近感が高まるのだという．これは心理学的には「相補性理論」というらしい．

相補性理論の究極は，DNAの選択だそうだ．人間のDNAは31億対あるが，人は知らず知らずにDNAの塩基配列が可能な限り自分と違う相手を選ぶ傾向があるのだという．これは種の保存のためには理にかなった行動だ．最近のNHKの番組で，学術的には世界的に有名だという老夫婦が紹介された．2人は遺伝子の組み合わせが考えられる範囲ですべて違うことがわかっているという．遺伝子の配列がすべて違う2人が出合う確率は，天文学的な数字になるらしい．その一方，絶滅が危ぶまれている世界最小の民族も紹介された．彼らは次の世代まで生き残れるかどうかすらわからないという．勇壮だった彼らは過去，多くの民族間の紛争を起こしてばかりいるうちに，いつのまにか世間から隔離された環境で暮らすことを選ぶことになった．DNAの選択肢を大幅に狭めた結果，原因不明の病気が続出するようになり，人口が急激に減少し，民族が絶えそうな状況になってしまっているということらしい．これは衝撃的だった．

● コンパチの世界

現在のコンピュータはすべてノイマン型であるという類似性をもつ．エッカート＆モークリからノイマン型の基本特許を買い取ったIBMが，同業のハネウェルなどに特許料の支払いを迫った裁判は（1967年），結局はアタナソフのABCコンピュータに起源をみとめ，敗訴となった．最大の障害をなくしたコンピュータ業界の発展は勢いづいた．1974年には日立・富士通がIBM互換機の発売を開始し，日本のコンピュータ産業の隆盛を現出させていった．富士通，日立は後にIBMに多額のロイヤリティを支払うことになるが，コンピュータ業界はそのくらい類似性を高めていたともいえる．パソコンで実現するIBM3270，富士通6650エミュレータ端末も，もちろん端末に類似性を求めたものだ．

そう，パソコンの世界も同じだ．初期こそ多様なマイコン・チップを使用した多くの種類が存在したパソコンの世界だが，このうちApple IIクローン，IBM PCクローンは格安で，付加価値もあって大成功した．これが，現在のWindows機，Macintoshの2大勢力の根源となったし，当時のクローン機製作メーカは，現在，マザーボード・メーカとして大活躍している．

ソフトウェアの世界では，たとえば，1979年に開発され，歴史に名を残したApple IIのVisicalcがある．後にCP/MのSuperCalc，MS-DOSのMultiPlan，Lotus-123，Excelと移り変わってきた．名前こそ違うが，それぞれ基本的な考え方はまったく同じで，極めて類似性の高いものになっている．

「似て非なるもの」――つまり「類似性」でユーザに親近感をもたせ，さらなる付加価値を付け加えた差別化を図った「相補性」でユーザの心を引きつけていった．そして，時に感情的，法律的な問題を起こしながらも和解，共存し，切磋琢磨することでコンピュータの驚異的な発展と普及へと導いた．

● マイクロソフトの成功の理由

マイクロソフトの歴史もまた，この例に漏れない．出発点となったBASICは，もともとダートマス大学のジョン・ケムニとトーマス・カーツが開発したオープン・ソースだった．これを，後にヒットしたマイコン・キットのアルティアにポーティングしたのがビル・ゲイツだ．これで成功した彼は1975年に，ポール・アレンとマイクロソフトを設立した．

ついで，当時一世を風靡していたCP/MにそっくりなQ-DOSをシアトル・コンピュータから買い取った．IBM PCのOSを



開発するためだ．機密保持契約でビル・ゲイツはIBMとの契約を話すことができず，格安の買い物となるが，これが後にMS-DOSに化けることとなる．

さらに，もともとはゼロックスのパロアルト研究所で開発され，アップルのLisa，そしてMacintoshに搭載されたGUIは，マイクロソフトではWindowsとして実現された．この件でマイクロソフトはゼロックス，アップルとの三つ巴の訴訟合戦に巻き込まれるが，後に和解し，最終的にはWindowsでもっとも成功した企業となる．

Webブラウザはイリノイ大学のマーク・アンドリューセンら4人が1983年に開発し，Netscape社を設立して世界的に普及させた．大きく遅参したマイクロソフトもInternet Explorerの大幅な機能強化とバンドル戦略を強力に推進し，徹底的に市場を制覇することに成功した．

そして最近，2004年3月には，Java搭載訴訟で，サンマイクロシステムズに7億ドル支払うことで和解したばかりだ．マイクロソフトの歴史はとりもなおさず,「似て非なるもの」の開発の連続だったといえるのかもしれない．

● 発展を拒むもの

最近，Windowsや，Offieceコンパチといわれる製品が多く出回っている．マイクロソフトはこれを誇りとするべきものだろう．しかし，そんな彼らは他社には，あまり寛容ではないように見て取れる．

Lindowsは，LinuxでWindowsの操作性を実現したものだ．マイクロソフトは，2001年12月に，Windowsの商標権使用で，小さな企業にすぎないLindowsを訴えた．通常，1文字違いの場合は商標権侵害が認められるが，Windowsのようにコンピュータ業界で20年以上も前から使用されている一般名称と思われる場合は，よほどのことでない限り商標権が認められることはない．また，マイクロソフトの製品名の一部を含んだソフトウェア製品は世界に数多く存在する．それらを訴えずにLindowsだけを商標権侵害で訴えるというのも苦しい気がする．

実際のところ，5月26日には，アメリカではマイクロソフトの上告が棄却され，連邦地裁に差し戻され，Lindows側が有利な状況だ．しかし，マイクロソフトは，全世界でLindowsを訴えており，各国でマイクロソフトと真っ向から戦うにはLindows



はあまりにも非力だ．多くの訴訟に対抗し，和解に達する可能性は極めて低いと判断し，Lindowsは名称を「Linspire」に変えて販売を続行することになった．

マイクロソフトの行動は，利益を追求する企業である以上，一概に否定できないかもしれない．しかし，小さな企業に対する仕打ちとしては，ちょっとばかり大人気ないような気もする．

類似性を認めることでマイクロソフトも切磋琢磨できるし，相補性を磨くことでユーザにも選択肢ができる．類似製品を倒し，自社だけが残っていくと，そのうち開発に行き詰ってしまうことはないのだろうか．コンピュータ業界のさらなる発展のためだけでなく，マイクロソフトのさらなる成長のためにも，もう少しおおらかでいてほしい．

勇壮だった民族が行き過ぎた自己防衛の結果，類似性の選択を狭め，みずからを破滅へと追い詰めていったという話を思い出してしまうのは，筆者だけだろうか．

あさひ・しょうすけ　テクニカル・ライター
イラスト　森 祐子

Engineering Life in

めざせIPO！

この原稿を書いている5月半ばのシリコンバレーでは，GoogleのIPO（Initial Public Offering）＝株式上場が大きな話題となっている．株式上場は，スタート・アップに勤める人達にとって大きな節目のイベントでもある．スタート・アップを選ぶ理由はさまざまだと思う．自分のやりたい仕事をとことんまでやれる環境だとか，仕事の内容が充実していたり，自由度が高い場合が多い．もっとも，一攫千金の夢を見てストック・オプションに賭けるエンジニア達が多いのもたしかである．

☆ インセンティブとなるストック・オプション

シリコンバレーのスタート・アップでもっとも多いストック・オプションとは，株式上場（株式公開）する前の自社株をある設定された価格で，決められた株数を購入できる権利である．株数は勤務年月によって分割されて配布される．これは，特に決まった期間はないが，多くは4年から5年ぐらいになる．社員に長く残ってもらうためのインセンティブの一つだ．

多くのスタート・アップは，会社のキャッシュ・フローをコントロールするために大幅に給料を低く設定しているところが多い．そのため，ストック・オプションで金銭的なインセンティブを与えるのは当たり前とも考えられる．例を挙げると，株は5年勤めてすべての数量を引き受ける権利が与えられ，初めの12か月後に初めの5分の1が与えられ，その後は1か月ごとに増えて行くシステムが多い．一般的には，株価は入社した前後の会社の評価額で決まる．

実際の例を出して説明するとこうなる．6,000株を一株$0.25で買えるストック・オプションとなると，初めの12か月後には，1,200株行使できる権利を与えられ，その後は1か月ごとに（6,000株－1,200株）÷（4年×12か月）＝100株与えられる．5年後満期になると5,000株すべて行使できる権利を引き受けることになる．買う権利を行使するには，6,000株×$0.25＝$1,500を会社に支払い，実際の株を買うことになる．最近は株券は発行されず，証券会社の口座を通じてすべての取引が行われる．その後，会社が上場した際，株の額面が$20になっていると，6,000×$20－$1,500＝$118,500，約1,000万円を丸儲けしたことになる．

スタート・アップが上場せず，ほかの会社に買収される例も存在する．同業者であるとか，同じ市場で存在する大手企業が買手になるわけだ．株式上場するには，市場関係者を納得させる材料が必要だが，スタート・アップで「研究開発のレベルは高いが，販路がまだ確立されていないケース」などの場合，大手企業に買収されたほうがさまざまな理由で有利な場合がある．この場合，ストック・オプションが買手企業の株に変換されるケースが多い．株を交換するほうが買収する企業のキャッシュ・フローには有利なケースが多いからだ．

☆ 株式市場で現金化

上場するか，買収されるか，いずれのケースも証券会社を通じて売却できる株…つまり流通されている一般公開株になってからが肝心だ．上場していない企業の株券やオプションは譲渡したり現金化することが非常に難しく，一般の人にとってこの手の取引は不可能に近い．よって，自社が上場して上場企業になり一般公開株が流通するか，自社株が上場している企業に買収されて一般公開株に交換されなければ意味がない．その後は，社員の判断でいつ売却・現金化を実行するかがポイントとなる．

もちろん株式市場により，一般投資家や機関投資家…株に投資する（買う）人…の興味をそそるような業績を上げたりすることが肝要となる．しかし，株式市場の動きは予想できないことが多いので，会社の努力だけではどうにもならないことが多い．　特にハイテク関連はまとめて評価されるので，あまり直接関係のない会社の業績が悪いと自分の会社の株にも影響がおよび，大幅な売り注文が入ると下落することがある．たとえばミドルウェアのソフトウェアの会社が上場しても，半導体企業の業績に株価がある程度影響されることがある．

☆ 健全なIPOはシリコンバレーの健康を示す

今回のGoogleのIPOに大きな興味があるのは，シリコンバレーの健康度を示すからだ．IPOが続かなければ新しい起業や新しいアイデア，そしてエンジニア達がシリコンバレーに集まらなくなるといえる．

シリコンバレーの大型IPOの連鎖作用は計り知れないほど大きい．投資家の興味レベルが高く，積極的なIPOだと，ほかの投資家もハイテク関連の銘柄に注目したり，市場のハイテクに対する態度を積極的な方向に変えるからだ．また，今回のGoogleのIPOが成功すると，ほかのシリコンバレー企業やハイテク企業でIPOを控えていた企業が再度IPOに向けて活動を復活させたりする．IPOに成功した会社は，集めた資金で新たな製品開発の大型投資を行ったり，必要な場合はほかのベンチャを買収するケースが多く，地元の雇用が増えたり，シリコンバレー外の拠点での増員が進んで多くのエンジニアが雇われることになる．ちなみに，今回のGoogleも集めた資金で企業を買収することが噂されている．

次に，ベンチャ・キャピタルやエンジェルなど，ベンチャ企業に初期のころにリスクを背負ってスポンサになった投資家達が資金を回収できる．ベンチャ・キャピタルもこの大型なキャピタル・ゲインを目的に投資を続ける．よって，その収益でまた新しいベンチャ企業を求めて投資を続け，ベンチャ企業に資金が潤う．一方，個人であるが，初期のころからベンチャ企業でがんばってきた社員も同じで，やっと金銭的な恩恵を受けられる．これで，もちろん個人消費も増え，シリコンバレーの地

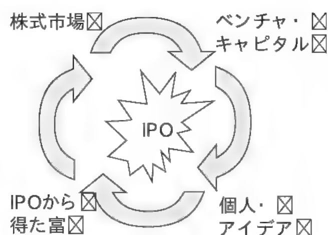


図1
シリコンバレーの起業サイクル

元景気に恩恵をもたらす．

また，多くの社員は「二匹目のどじょう」を求めて違うスタート・アップに行くなり，自分で起業を試み，スタート・アップを起す人が多い．もっと大金持ちになった個人は，自分でエンジェルのような個人投資家になるケースもある．これらの「二匹目のどじょう」を狙う人達は，エンジニアや技術職に就く人以外にもいえる．スタート・アップで実務経験を積んだ人達は，一度リスクを背負って仕事をしてきているし，だいぶ度胸ができる．特に初期のころから仕事をしてきた人達はいろいろな仕事を兼務することが多いのでかなり内容の濃い経験を積んでいることになり，ネットワークも広い場合が多い．特にベンチャ企業の設立のノウハウやコネなどはスタート・アップでの仕事によって手っ取り早く身に付けられる．このようにIPOを通じて起業のサイクルが一回りして元に戻ることになり，新たな起業へつながるチャンスが広がる．これにより，さらに投資基金が集まり，アイデアを持った人達もシリコンバレーを拠点に起業を試みようとする．

つまり，IPOはシリコンバレーの起業サイクルを完結させる起爆剤であり，シリコンバレーでの新陳代謝を促す重要な役割を果たしている（図1）．

伝説になってしまった話だが，ストック・オプションのことをあまり良く知らなかった社員が，ボーナスや昇給の代わりにストック・オプションをもらい続け，会社を引退するころにすべてのオプションを行使したら$100万以上になっていたという話がある．この社員がエンジニアとか実務ではなく，受け付けとか用務員のオジサンみたいな仕事の場合にはもっと衝撃的なニュースとなる．この手の話を聞くと，だれでも一攫千金の夢をつかむことができるのでは？と思ってしまう．エンジニアや上層部の人間でなくてもストック・オプションの恩恵を受けられる一種のアメリカン・ドリームのチャンスとしてシリコンバレーの夢はいいなぁ〜と，みんな思うのである．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

Column IPOのしくみ

株式上場（IPO）は一般投資家と接する初めての機会となり，会社が公の場でデビューする舞台でもある．上場前の企業は財務や業績情報の公開が義務付けられていなく，シリコンバレーのスタート・アップの初期は，数四半期に渡って赤字を出すのが当たり前だ．しかし，株式が公開された時点で市場関係者に細かく説明をする義務が発生する．

株式上場のプロセスは複雑で，多くの金融や法律の専門家の手助けが必要となる．まずはSEC，アメリカ証券取引委員会に上場する主旨の書類を作成し提出するが，これには会社の財務状況や売り上げ状況を細かく報告した監査報告の提出が必要になる．

実際の株式市場と会社の間を取り持つのは，インベストメント・バンク（投資銀行）と呼ばれるタイプの銀行になる．投資銀行はIPO全体の約20%を手数料として受け取るので，銀行側としては大きな収入源でもある．IPOを行うためには，最低$2,500万の年間売り上げが必要だ．会社の評価額はさまざまな算出方法があり，投資銀行が中心になって算出を行う．目安は最低年間売り上げの2倍から3倍がスタート額となる．つまり$3,000万年間売上げの会社だと，$6,000万〜$9,000万程度の評価額となる．多くの場合は，IPOで総株数の20〜40%を市場に売り出すことになり，額面が通常のケースだと$10から$20の間が好まれるので評価額と額面に応じて市場に売り出す株数を調整する必要がある．また，競合や同業者がすでに市場に存在する場合，その会社のPEレシオを参考に会社の評価額を算出する場合が多い．いずれにせよ，18か月から24か月きれいな右上がりのカーブで売り上げが伸びていなければIPOの候補にはなれない．

書類や届出が整うと，新しく発行される会社のProspectusと呼ばれる趣意書が作成され，これには細かく会社のビジネスやリスクについて，素人やハイテクの門外漢でもわかるように説明している．株主訴訟を起されては困るのでかなり細かく書かれている．このProspectusと共に投資銀行は，自社のコネのある機関投資家などへ売り込みを行い，購入の興味レベルを測る．この売り込みは，アメリカの大都市で行われたり，必要な場合はロンドンなど主要なヨーロッパ都市でも行われ，ロードショーと呼ばれる．ロードショーが終わると最終的なProspectusが発行され，SECに正式に提出され実際に株式市場で株の取引が行われる．投資銀行側で大型顧客に割り当てが決まっているので，証券会社に相当なコネがないと人気のあるIPOに個人でオープンの日に参加することは難しい．インターネット・バブル時には，これが不正や不祥事につながったりしたので，いまだにIPOや大手投資銀行を警戒する投資家も多い．今回のGoogleのIPOは投資銀行を通さず，会社側でインターネットを使った入札を行って株価と発行部数を決めるというユニークな方法を取っている．Google側では，この新しいシステムにより公正なIPOをめざしたいとしている．個人でも$2,000さえ口座に入金すれば参加できるシステムになっており，このユニークなシステムも今回のIPOの興味をそそる一つの大きな理由となっている．

●32ビット・プロセッサ

Nios IIファミリ

- ソフト・コアの32ビット・エンベデッド・プロセッサで，200DMIPS（Dhrystone MIPS）以上の性能を提供し，FPGA製品への実装を低コストで実現．
- ソフト・コアなので，システム・レベル設計にASICの使用を検討する必要がない．
- 無限の組み合わせのシステム・コンフィグレーションの中から，性能およびコスト目標に合うものを選択できる．
- システム性能を最大限に引き出すコア，最小ロジックの使用に最適化されたコア，およびこの2種類のバランスをとるコアの3種類のソフトCPUコアから構成されており，すべてのコアは完全にコード互換．
- コンパイラ，統合開発環境（IDE），JTAGデバッガ，RTOSおよびTCP/IPスタックを含むソフトウェア開発群を装備．

●価格：下記へ問い合わせ

■日本アルテラ（株）
TEL：03-3340-9415 FAX：03-3340-9487

●プロセッサ・コア

Xtensa LX プロセッサ・コア

- あるクロック・サイクルで使用されていないロジック部分への電力供給を行わないという，クロック・ゲーティングを自動的に行い，低消費電力を実現．
- オプションである第2のロード/ストア・ユニットと，ユーザ定義のポートおよびキューの二つの技術革新により，Tbpsレベルでデータの入出力が可能なI/Oスループットを実現．
- 一つあるいは二つの128ビット幅のロード/ストア・ユニットの選択が可能．
- 演算処理能力は，Xtensa命令セット・アーキテクチャを効率化した，FLIXアーキテクチャにより実現．
- メモリ・アクセス速度によるプロセッサの処理速度低下に対応するため，コンフィギュラブル・パイプラインをサポートすることでオンチップ・メモリに対するインターフェースを改良．

●ライセンス価格：$600,000～

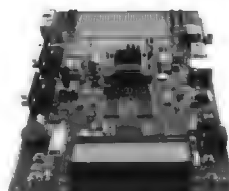
■テンシリカ（株）
TEL：045-477-3373

●16/32ビット・マイコン

STR710シリーズ/STR720シリーズ

- STR710シリーズは，ARM7TDMIコアをベースとしており，組み込みフラッシュ・メモリやロー・ピン・カウント・パッケージなどを装備．32ビット・マイコンのパワーと柔軟性に加え，豊富なオンチップ・ペリフェラル・セットを必要とする製品に適する．
- STR720シリーズは，ARM720Tコアをベースとしており，キャッシュ，MMU，SDRAMインターフェースを装備．パフォーマンスと先進のオペレーティング・システムのサポートが容易なオープン・システム・メモリ・アーキテクチャを必要とするアプリケーションに適する．

●サンプル価格：STR710Z2 ¥800（20個時）
STR720RB ¥1,000（20個時）



■STマイクロエレクトロニクス（株）
TEL：03-5783-8340 FAX：03-5783-8216

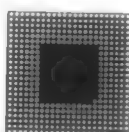
●アプリケーション・プロセッサ

SH-Mobile3

- 新規開発のCPUコア「SH4AL-DSP」を搭載し，最大動作周波数216MHz時に389MIPSの処理性能をもつ．
- 負荷の大きな複数アプリケーションの並列処理や，専用OSより処理負荷が大きなLinuxなどの汎用OSによる動作などの実現が可能．
- 「SH4AL-DSP」の命令セットは，従来製品のCPUコア「SH3-DSP」と上位互換であり，既存プログラムの流用ができるため，システムの開発期間の短縮が可能．
- 300万画素クラスのカメラ・モジュールと直結可能なカメラ・インターフェースを搭載．
- 高精細カメラの大容量画像データを高速に取り込み，画像の電子ズーム表示，OSD機能やHWC機能などによる，画面の重ね表示などの多彩な表示が可能．

●価格：¥4,000（10,000個時）





■（株）ルネサステクノロジ
TEL：03-5201-5234

●32ビット・プロセッサ

S1C33L05

- ポータブル電子辞書向けに，カラーSTNコントローラ，USBフル・スピード（FS）ペリフェラル，スマートメディア・インターフェース，SDRAMインターフェース，NAND型フラッシュ・メモリ・インターフェースを1チップに集積．
- 高コード効率を実現する命令セット，積和演算機能をもち，音声や画像などのマルチメディア処理に適したプロセッサ．
- 微細プロセスの採用と同社の超低リーク技術により，消費電力を従来品と比較し約1/5に低減．
- カラーSTN専用のLCDコントローラを搭載しており，内蔵の40KバイトVRAMを用いて，1チップでQVGA 16色パッシブ・パネルを駆動することが可能．
- 外部SDRAMを接続することにより，最大QVGA（65,536色）のカラーSTNパネルを駆動することが可能．

●サンプル価格：¥1,700（QFPパッケージ）
¥1,200（ベア・チップ）

■セイコーエプソン（株）
TEL：042-587-7665
URL：http://www.epsondevice.com/

●小型16ビット・マイコン

M16C/Tinyシリーズ

- フラッシュ・メモリおよび車載LAN規格のCANに対応するコントローラを内蔵したM16C/29グループ．
- CAN対応コントローラを1チャネル内蔵．
- プログラム格納用途のフラッシュ・メモリを最大128Kバイトに拡大し，プログラム規模の増加に対応できる．
- RAMを最大12Kバイトに拡大し，CPUの処理性能向上が図れる．
- プログラムおよびデータ格納用途に使用可能な4Kバイトのフラッシュ・メモリを搭載．
- CPUコアは，16ビットの「M16C/60」を搭載し，動作電圧が3.0～5.5Vの範囲で，最大動作周波数は20MHz，最小命令実行時間が50nsという高速動作を実現．

●サンプル価格：¥860～¥960





■（株）ルネサステクノロジ
TEL：03-5201-5235
URL：http://www.renesas.com/jp/m16ctiny

●DSP

TMS320C6410/ TMS320C6413

- 「TMS320C6410」は動作クロック400MHzでメモリ容量が合計160Kバイト，「TMS320C6413」は動作クロック500MHzでメモリ容量が合計288KバイトのDSP.
- マルチチャネル・クロック・シリアル通信をサポートする標準McBSP×2，IISおよび主要なステレオCODECと互換性を持つオーディオ・シリアル・ポート×2，16/32ビットのマイクロプロセッサ・ホスト・ポート・インターフェース，IIC制御シリアル・ポート×2などをオンチップで搭載.
- DSP設計を迅速に行うために，「C6413 EVM」（評価モジュール）および「TMS320C6416 DSK」DSPスタータ・キットの2種類の開発ボードを供給.
- いずれのDSPも「Code Composer Studio」IDEを始めとする，同社の「eXpressDSP」ソフトウェアおよび開発ツールでサポートされる.

●サンプル価格：
TMS320C6410 $17.95（10,000個時）
TMS320C6413 $28.95（10,000個時）

■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036
URL：http://www.tij.co.jp/pic/

●ファイバ・チャネル・プロトコルIC

HPFC-5700A Tachyon DX4+

- ストレージ・サブシステムやストレージ・ルータ，ホスト・バス・アダプタやホスト・コンピュータのマザーボードなど，マルチプロセッサ・システムを使用しているミッド・エンドからハイ・エンドにいたるストレージ装置の性能を拡張することが可能.
- 既存の設計資産を再利用しながら，4Gビット/sの高速ファイバ・チャネル伝送へのアップグレードが可能.
- 2チャネルのファイバ・チャネル用デバイスにおける完全なハードウェア・ベースのソリューションであり，データ・インテグリティ・フィールド（DIF）機能を内蔵.
- サンプル・ドライバを含む，豊富なAPIツール・セットを提供.

●サンプル価格：下記へ問い合わせ



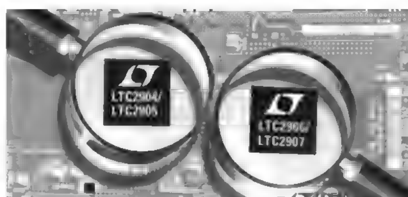
■アジレント・テクノロジー（株）
TEL：0120-61-1280
URL：http://www.agilent.co.jp/

●デュアル電源モニタ

LTC2904/LTC2905, LTC2906/LTC2907

- LTC2904/LTC2905は，5/3.3/2.5/1.8/1.5/1.2/1Vの9種類の組み合わせのスレッショルドを設定できる.
- LTC2906/LTC2907は，5/3.3/2.5V電源向け固定スレッショルド選択と，調整可能な低電圧入力を装備.
- LTC2905，LTC2907は，リセット遅延時間の調整が可能.
- 5%，7.5%，10%の三つの電源許容誤差の選択が可能.
- 保証スレッショルド精度は，全温度範囲において±1.5%.
- グリッチに対する耐性を備えているため，誤ったトリガリングのない信頼できるリセット動作が可能.

●サンプル価格：¥153～（1,000個時）



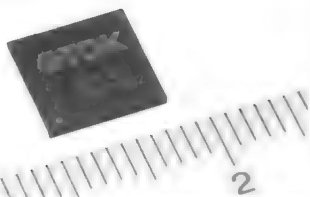
■リニアテクノロジー（株）
TEL：03-5226-7291 FAX：03-5226-0268

●フラッシュ・メモリ・コントローラ

GBDriver XR

- 64Mから16GビットまでのNAND型フラッシュ・メモリに対応した，フラッシュ・メモリ・コントローラIC.
- ホスト側の物理インターフェースとして，汎用SRAMバスのバースト・モード，ランダム・アクセス・モードに対応.
- システム設計側にIDEチャネルをサポートしていないLSIを採用した場合でも，大容量，低コストのNAND型フラッシュ・メモリの使用が可能.
- CompactFlashカード，シリコン・ディスク・モジュール用途のコントローラと同じフラッシュ制御アーキテクチャを採用しているため，バースト書き込みで5Mバイト/sを実現.

●サンプル価格：¥1,000



■TDK（株）
TEL：047-378-9232

●Hi-speed Link System用センタIC

MKY36

- 2,000点以上の信号のすべてを，1msの応答能力で収集，配信を可能にする.
- 信号点数が少ない場合は，0.1msの応答能力を発揮することも可能.
- 5V系のTTL信号に接続でき，3.3V動作にも対応.
- HUBへの対応を可能とすることで，割り込み発生などの機能性を向上.
- 本体にRAMを内蔵.
- 0.5mmピッチの64ピンQFP（12×12mm）で提供.
- 郵便仕分け自動化システム，半導体製造装置，高度医療機器，工業用ロボット，工作機械などの内部装置に適する.

●価格：下記へ問い合わせ



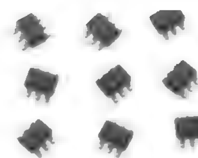
■（株）ステップテクニカ
TEL：04-2964-8804 FAX：04-2964-7653
URL：http://www.steptechnica.com/

●RFIC

ABAシリーズ ブロードバンド・シリコンRFIC

- ディジタル衛星放送受信機のLNB，ケーブル・テレビ，ケーブル・モデム，携帯電話基地局，ミリ波通信システム，光ファイバ通信システムなどの通信アプリケーションのIFアンプやバッファ・アンプに適する.
- 優れた低雑音指数に加え，＋2.2～＋16.1dBmのリニア出力パワーと，＋13.1～＋27.8dBmの出力3次インターセプト・ポイントを実現.
- ABA-31563/32563は，単一3Vの電源電圧で動作する低消費電力を実現.
- ABA-32563はDC～2.5GHz，ABA-31563はDC～3.5GHz，ABA-54563はDC～3.4GHzまでの広帯域で，良好なゲイン平坦性を有している.

●サンプル価格：¥35（ABA-31563）
¥40（ABA-32563）
¥50（ABA-54563）



■アジレント・テクノロジー（株）
TEL：0120-61-1280

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

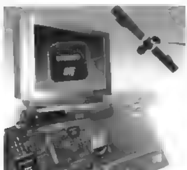
NEWPRODUCTS

●1チップ・シリコン・チューナ

STB6000

- ディジタルTVやWebベースのコンテンツを，衛星経由で受信するためにSTBで利用されている，ディスクリート・コンポーネントのチューナに代わるものとして設計されている．
- スタンドアロンの機能ブロックでだが，同社のSTV0299マルチスタンダード・ディモジュレータとともに，統合チップ・セットの一部として機能する．
- シンプルな2ワイヤ・シリアル・インターフェースを通じて，プログラミングされている．
- DVB（ディジタル・ビデオ・ブロードキャスティング），DirecTV，VSAT（超小型衛星通信基地局）の各プロトコルをサポートしているため，全世界の衛星用STBでの利用が可能．

●サンプル価格：¥2,000（10個時）



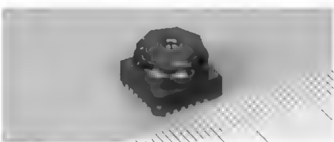
■STマイクロエレクトロニクス（株）
TEL：03-5783-8340 FAX：03-5783-8216

●カメラ・モジュール

FPDF0Xシリーズ/FPDF8Xシリーズ

- 「FPDF0Xシリーズ」は1.3Mピクセル，「FPDF8Xシリーズ」は1.3Mピクセルでオート・フォーカス機能を搭載．
- カメラ部で，固有の光学設計と非球面ガラス・レンズ技術により，F2.8を実現した高精細レンズを搭載．
- オート・フォーカス機能では，独自の切削加工技術，金型加工技術を駆使して開発したステッピング・モータにより，レンズ移動ピッチ11.25μm，ステップ数35ポジションを達成．
- 内蔵された画像処理用DSPとアルゴリズムにより，リニアなオート・フォーカスが可能となり，接写から人物，風景撮影までスムーズな操作が可能．

●サンプル価格：
¥15,000（FPDF0Xシリーズ）
¥25,000（FPDF8Xシリーズ）



■アルプス電気（株）
TEL：03-5499-8154

●Bluetoothプロトコル・アナライザ

BAP-D12

- シンガポールのMobiwave社が開発した，Bluetoothプロトコル・アナライザ．
- スニッフ，ホールド，パーク，パーク（PM ADDR）のすべての省電力モードとロール・スイッチをサポート．
- リアルタイム・キャプチャ，デコード，暗号複合化，検索などの機能で，パワー・マネージメントのインプリメンテーションの動作を捕捉可能．
- 解析ツール「Smart Tandem」は，複数のBAP-D12ハードウェアが，リアルタイムでキャプチャしたデータを統合し，完全なピコネットの全容を再現する機能をもつ．
- EV3，EV4，EV5のeSCOパケットを完全にサポート．
- eSCOのオーディオ・ストリームを再構成する機能により，WAVEファイルでの再生，CVSDフォーマットまたはPCMフォーマットへのエクスポートが可能．

●価格：下記へ問い合わせ

■エボンエレクトロニクス（株）
TEL：03-5600-5770 FAX：03-5600-8775
E-mail：info@evon-ele.co.jp
URL：http://www.evon-ele.co.jp/

●ディジタル・オシロスコープ

DL1740E/1740EL

- DL1740ELは最大8Mワード，DL1740Eは最大2Mワードで，従来機と比較して，8倍のロング・メモリを搭載．
- サンプリング・スピードを1Gspsに維持したまま，8msの間データの捕捉が可能．
- 時間軸の設定を長くしても，高速ノイズなどを確実に観測することが可能．
- USBインターフェースを経由して，フラッシュ・メモリ，ハードディスク・ドライブ，MOドライブなどとの接続が可能で，波形データや測定画面のイメージを保存することができる．

●価格：¥1,280,000（DL1740EL）
¥980,000（DL1740E）



■横河電機（株）
TEL：0120-137-046 FAX：0422-52-6624

●無線LANアクセス・ポイント

AP-5100A，AP-50

- AP-5100Aは，IEEE802.11a/g同時通信対応の，ルータ機能付き無線LANアクセス・ポイント．タグVLAN機能により，最大17の仮想LANの構築が可能．高速ルーティング機能に対応したブロードバンド・ルータとして，光ファイバ，ADSL，CATV回線の環境下での使用が可能．
- AP-50は，手のひらサイズ（120×103×29mm）の無線LANアクセス・ポイント．切り替え方式で，IEEE802.11a/b/gに対応．
- 無線LANの高速化技術Super A/Gに対応し，データを圧縮して送信することが可能．
- IEEE802.1x/EAP，OCB AES，WPA-PSKの高度なセキュリティ機能に対応．
- PoE対応により，LANケーブルで電流供給が可能．

●価格：オープン価格

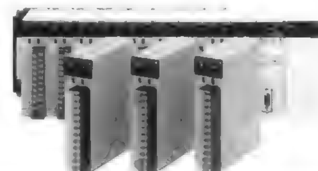
■アイコム（株）
TEL：06-6792-4949

●プロセス入力ユニット

形CS1W-PTS55，形CS1W-PTS56，形CS1W-PDC56

- 温度管理，モニタリング用に温度センサを直接接続可能な「熱電対入力ユニット（形CS1W-PTS55）」，「測温抵抗体入力ユニット（形CS1W-PTS56）」，流量・圧力・レベルなどのセンサからのアナログ信号を4～20mA，1～5Vなどのレンジで取り込み可能な「直流対入力ユニット（形CS1W-PDC56）」の3種類をリリース．
- CSシリーズ1スロット・サイズでありながら，チャネル間絶縁機能を取り込み，1ユニットあたり8点の入力が可能．
- PLC計装コンセプト「SMARTPROCESS」により，開平演算機能，オンディレイ，ヒステリシス付きの警報機能など，プログラミング工数を削減する機能を搭載．

●価格：¥120,000



■オムロン（株）
TEL：055-977-9024

●LCDモニタ

兼松KSMシリーズ 機器組み込み用LCDモニタ

- アナログRGB(PC信号)およびコンポジット(映像信号)に対応し，音声回路(ステレオ)を搭載.
- 国内および海外の主要パネル・メーカのVGA～SXGAパネルに対応.
- タッチ・パネル出力は，USBまたはRS-232-Cの選択が可能.
- 機器操作用タッチ・パネルの有無を選択可能.
- 標準品でありながら，多彩なバリエーションに対応.
- 信号に入力がない場合，一定時間を経過すると省電力モードに移行.
- オプションで，近接センサにより人体が近づいたり離れた場合，モニタの電源をON/OFFする機能を搭載できる.

●価格：¥64,000～¥86,000



■兼松（株）
TEL：03-3544-6533 FAX：03-5565-0093

●開発プラットホーム

ST7MC Starter-Kit

- モータ制御アプリケーションをターゲットとし，BLDCなどの3相ブラシレス・モータとインダクション・モータを使用するアプリケーションの市場投入までの時間を短縮し，開発コストを低減するST7MCマイクロコントローラ開発プラットホーム.
- クイック・デモと評価の両機能を提供するほか，包括的なハードウェアは，コンセッションなしのソフトウェア・パッケージとともに，プロジェクトの高度な段階までのデバッグが可能.
- ハードウェア・プラットホームは，センサレスBLDC送風機，インバータ制御ボード，高電圧絶縁ボード，および通信，プログラミング，デバッグ機能用のSofTec inDART-STXデバッガ，プログラマで構成され，USBインターフェースを装備.
- ソフトウェアは，GUIとC言語対応ソフトウェア・ライブラリ・セットを装備.
- 完全なシステムはPC接続を目的としているが，スタンドアロン・モードでも稼動.

●サンプル価格：¥80,000

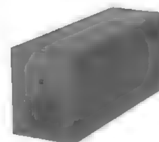
■STマイクロエレクトロニクス（株）
TEL：03-5783-8240 FAX：03-5783-8216

●拡張シャーシ

ECH(PCI)BE-***シリーズ，EAD(**)BEシリーズ

- ECH(PCI)BE-***シリーズはPCIバス拡張シャーシで，EAD(**)BEシリーズは拡張バス・アダプタ．パソコンの拡張スロット仕様と必要なPCIスロット数，ボード・サイズに応じて，19通りのシャーシとバス・アダプタの組み合わせが可能.
- ECH(PCI)BE-H2B/F2Bは2スロット，ECH(PCI)BE-H4B/F4Bは4スロット増設可能.
- ECH(PCI)BE-H2B/H4Bはショート・サイズのPCIバス・ボード，ECH(PCI)BE-F2B/F4Bはロング・サイズのPCIバス・ボードを実装可能.
- EAD(**)BEシリーズでは，必要なPCIバスのスロット数，ボード・サイズに合わせて拡張シャーシを選択することが可能.

●価格：ECH(PCI)BE-***シリーズ ¥57,750～¥115,500
EAD(**)BEシリーズ ¥28,350～¥29,400



■（株）コンテック
TEL：03-5628-9286 FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●DV伝送システム

CamOnIP on PC-CUBE

- 手のひらサイズの小型装置に，DVカメラをi.Linkで接続し，Ethernetに接続するだけで，テレビの品質フレーム，フルサイズの高品位な動画像を伝送することが可能.
- 帯域の狭い回線を使用する場合には，フレームを間引き，占有帯域を削減する.
- 音声データは，間引くことなく通常の会話を可能とする手法を採用.
- 広域IP網でも，高品位なテレビ会議を行うことが可能.
- 電源を入れるだけの簡単な操作で，通信を行うことができる.
- CPUにPentium M 1.1GHzを採用し，マルチメディアに特化したテクノロジを採用.
- i.Linkケーブルを接続するだけで，自動的にカメラを認識して通信を開始する.

●価格：オープン価格

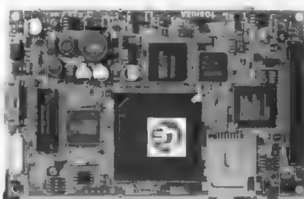
■FAシステムエンジニアリング（株）
TEL：03-3472-0017 FAX：03-3472-0018
E-mail：faseinfo@fase.co.jp
URL：http://www.fase.co.jp/

●開発キット

T-Engine/TX4956 開発キット

- MIPSアーキテクチャをベースに，東芝が開発した64ビット・プロセッサ「TX49/H4コア」を搭載.
- 回路設計の最適化により，400MHzの内部動作周波数で，消費電力0.6Wを実現.
- 多機能プリンタやカーナビ，STB，ハードディスクなどのディジタル機器の開発に適している.
- 標準T-Engineボードのほか，同社のRTOS「PMC T-Engine」，開発用基本ミドルウェア，GNU開発環境，仕様書などのドキュメント類が含まれる.

●価格：¥207,900



■パーソナルメディア（株）
TEL：03-5702-7858
E-mail：te-sales@personal-media.co.jp
URL：http://www.personal-media.co.jp/te/

●計測ステーション

WE500/WE900 計測ステーション

- 同社のPCベース計測器「WE7000」シリーズ用の，計測モジュール格納ベース・ユニット.
- CPUの性能を向上させ，計測ステーションにデータ処理機能を追加することで，データ処理を高速化.
- PCを使用しない，自律処理システムの構築が可能.
- 本体にUSBインターフェースを装備したことで，PCと計測ステーションの通信設定が不要.
- Ethernetインターフェースを，モジュール装備から本体装備に改善.
- モジュールを挿入するスロット数を，5スロットおよび9スロットに拡張.
- 電圧，電流，波形，温度，加速度，回転，振動，変位などの汎用計測に適する.

●価格：¥300,000(WE500)
¥400,000(WE900)

■横河電機（株）
TEL：0422-52-6613 FAX：0422-52-6624

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX:(03)5395-2127，E-mail：mngnews@cqpub.co.jp (編集部)

●産業用パソコン

FAPC-1000

- CPUにPentium4 2.4GHzを採用し，メモリは標準で1Gバイト搭載.
- 対応OSは，LinuxおよびWindows XP Professional/NT Workstationをサポートし，別途契約でEmbeddedにも対応.
- オプション設定として，FDD，CD-R/RW，MO，HDD，ホットスワップ対応のミラーリングHDD，メンテナンス・フリーのコンパクト・フラッシュ・メモリの搭載が可能.
- 24時間稼動のRAS機能，個別仕様など用途，目的に合わせた構築が可能.
- 独自の生産方式で，7年間の販売供給，生産完了後3年間のメンテナンスを保証.

●価格：¥1,200,000～



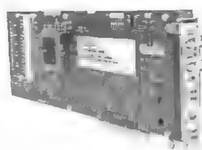
■日本ミニコンピュータシステム（株）
TEL：0422-28-4100

●PCIバス用A-D変換ボード

CompuScope 14200

- 14ビット分解能，200Mspsでバンド幅が100MHzの，PCIバス用A-D変換ボード.
- 最大1Gバイトのメモリを搭載可能で，2チャネルの同期高速データ収録が可能.
- DCオフセットの制御，外部クロックとトリガ信号の入出力を標準装備しているため，外部機器との同期が容易にとれる.
- 32ビット，66MHzのPCIバスにより，200Mバイト/sの速度でPCへのデータ転送が可能.
- 豊富なサンプル・プログラムで構成された，C/C++，LabVIEW，MATLABなど各種言語用開発キットにより，アプリケーションの開発が容易.
- Windows NT/2000/XPに対応.

●価格：下記へ問い合わせ



■（株）ロッキー
TEL：03-3228-4511 FAX：03-3388-1391
URL：http://www.kkrocky.com/

●半導体設計・検証ツール

Synplify DSP

- MathWorksの「Simulink」ユーザは，アルゴリズム・レベルで書かれたデザインから，高品質で合成可能なRTLレコードの自動生成が可能.
- システムレベル・リタイミングなどのアルゴリズムを用いて，RTLコード生成前に「Simulink」ベースのデザインを，システム・レベルで最適化.
- 自動マルチ・チャネライゼーション技術により，シングル・チャネルのスペックから，マルチ・チャネルのシステムを自動的に生成し，スレッド容量の「What-if」分析を迅速に実行することが可能.
- 性能とエリアのトレードオフを実行する，独自のフォールディング・アルゴリズムを使用.

●価格：¥3,780,000～

■シンプリシティ（株）
TEL：03-5358-3311

●システム監視・復旧ツール

WinPatrol

- Windowsシステムとは別のRTOSをベースとしているため，客観的にWindowsの稼動状況を監視でき，Windowsの障害時にも，影響を受けることなく障害の検知および通知が可能.
- システムの運用監視に必要な「監視」，「警報・通知」，「保守支援」，「制御」の4種類の機能を装備.
- 障害を検出した際には，プロセスおよびシステムの安全性を考慮したシステムの停止，再スタートなどの処理や，メンテナンスに必要な情報収集を行う.
- メールやシリアル出力を始めとする各種警報や通知機能により，オペレータやシステム管理者に障害の発生を通知することが可能.
- システム全体に影響を与える，Windowsシステムの異常な負荷状態を検出.
- 全プロセスのメモリ利用状態を監視し，長時間の稼動によって起こりうるメモリ・リークの傾向を検出し，問題発生を未然に防止.

●価格：¥58,000

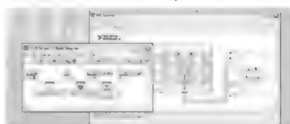
■（株）アスキーソリューションズ
TEL：03-4524-6002
E-mail：server@asciisolutions.com
URL：http://www.asciisolutions.com/

●計測プラットホーム

SC-2350，BNC-2096，SCXI-1314T

- SC-2350は，NI SCC信号調節モジュール用キャリアで，既存のSCCアナログ入力モジュールすべてに対応し，TEDSリーダを内蔵.
- BNC-2096は，TEDS搭載加速度計およびマイクロフォン用の19インチ・ラック・マウント端子台で，PXI-4461 24ビット・アナログ入出力データ集積モジュール，PXI-4472 8チャネルDSAモジュールおよび，SCXI-1530/1 4チャネル加速度計入力モジュールなどのIEPE信号調節用のフロント・エンドとして機能する.
- SCXI-1314Tは，SCXI-1520ブリッジセンサ入力モジュール用のフロント・マウント端子台で，ブリッジ・ベースのロード・セルや圧力センサのTEDS情報を読み取ることを，SCXIプラットホーム上で実現.

●価格：SC-2350 ¥55,000
BNC-2096 ¥83,000
SCXI-1314T ¥45,000



■日本ナショナルインスツルメンツ（株）
TEL：0120-527196 FAX：03-5472-2977
E-mail：prjapan@ni.com

●RTOS/開発ツール

Nucleus RTOS，開発ツール

- アルテラ・コーポレーション社製のNios IIエンベデッド・プロセッサ・ファミリ向け，RTOSおよび開発ツール.
- Nios IIプロセッサを使用したネットワーキング，テレコム，車載用，ディジタル・エンターテインメント，産業用制御機器などの組み込みアプリケーション構築の全過程を終始一貫して支援する，ロイヤリティ不要のトータル・ソリューションを提供.
- Nucleus PLUSリアルタイムカーネル，code|lab EDE統合開発環境およびcode|lab Debugソースレベル・デバッガは，Nios IIソフトコア・エンベデッド・プロセッサ・ファミリに向けた，アクセラレイテッド・テクノロジによる，組み込み製品ラインオファーの主軸となる.
- Nucleus PLUSは，効率的でスケーラブルな高速軽量リアルタイム・カーネルで，ソース・コードとともに提供.

●価格：下記へ問い合わせ

■メンター・グラフィックス・ジャパン（株）
TEL：03-5488-3041 FAX：03-5488-3032
E-mail：jpn_press@acceleratedtechnology.com

●開発支援システム

Duepark

- インターネットを利用した，ソフトウェア品質管理システム．
- 複数の企業や開発者が，共通のソフトウェア品質メトリクスの下で開発でき，要求される品質水準を全工程で維持できる．
- 定量化が難しいといわれていた開発中のソフトウェア品質を，p-u管理図，バグ分類傾向分析，テスト実績記録，信頼度成長曲線，品質メジャーなどのツールで可視化．
- 品質管理業務に伴う専門知識とそれに基づくデータ収集や分析をすべて自動化．
- 品質管理支援ツール5種類に加え，工程レビュ記録，ステップ・カウンタ，直交表，機能マトリクスなどのソフトウェア開発に必要なツールを装備．
- インターネット上の仮想共有空間を利用し，完全なリアルタイム処理で，遠隔地同士の開発者がアプリケーションを利用できるオプションを用意．

●価格：¥1,200,000～

■オムロン ソフトウェア（株）
TEL：044-246-6001 FAX：044-246-6012

●ソフトウェア共同開発ソリューション

SourceForge 3.4 Enterprise Edition

- ソフトウェアのチーム開発を効率的に促進，管理するためのタスク管理，文書管理，トラッカ機能などのツールを統合した環境を提供．
- 対応ブラウザによって，どこからでもプロジェクトの最新リソースにアクセス可能．
- 各プロジェクトの進捗状況をグラフや表を使って，直感的に把握．
- タスクに対して自動通知設定ができるため，問題を早期に発見でき，深刻化する前に対処が可能．
- 不具合情報，変更要求，文書ファイル，メーリング・リストのログ，ソース・コードなど，ソフトウェアの開発過程で発生する情報の一元管理が可能．
- 商用のソフトウェア構成管理ツールやプロジェクト管理ツールと連携できるほか，独自ツール間の連携が可能．
- プロジェクト・メンバに適切なアクセス権限を機能ごとに設定することが可能．

●価格：¥198,000/1ユーザ

■VA Linux Systems Japan（株）
TEL：03-3293-5151 FAX：03-3293-5152
E-mail：sf-sales@valinux.co.jp

●SNMPエージェント・プロトコル

PrSNMP

- 高速TCP/IPプロトコル・スタック「PrCONNECT2」のアドオン製品で，組み合わせで使用することで，SNMPエージェント機能を組み込んだネットワーク機器の開発が容易になる．
- 組み込み機器にSNMPエージェントの機能を，短期間で実装するために開発されたSNMPバージョン1に対応．
- 現在の生産情報やエラー情報など機器独自の情報を，プライベートMIBとして収集しておくことで，リモート監視を可能にする．
- エラーや状態遷移が発生した場合などにトラップを発行することで，管理者へのリアルタイムな報告を可能にする．
- MIB-II（RFC1213）を標準でサポートし，UDP/IP上で動作．
- ASN.1の基本符号化ルールによる，MIBオブジェクトのデコード，エンコードをサポート．

●価格：下記へ問い合わせ

■イーソル（株）
TEL：03-5302-1360 FAX：03-5302-1361
E-mail：ep-inq@esol.co.jp
URL：http://www.esol.co.jp/embedded/

●端末エミュレータ

Reflectionシリーズ Windows版v12/Web版v7

- 一括管理・構成，メータリングの各機能，さらにほかの最新技術との親和性によって，ITの柔軟性と管理性を最大化する端末エミュレータ．
- 多様なホスト・アプリケーションへの安全で，信頼性が高く管理しやすいアクセスを提供．
- 重要な企業データや基幹情報を確実に保護するために，SSL/TLS，Secure Shell，Kerberosなどの高度なセキュリティ機能を搭載．
- カスタマイズ・オプションで，ユーザの多様で特殊なニーズに応え，生産性を高めることが可能．
- JavaベースのAPIやMicrosoft VBA6.4に対応．
- IBMメインフレーム，IBM AS/400，OpenVMS，HP e3000，HP 9000，Tandem，Unisysを含む幅広いホスト・システムに対応．

●価格：下記へ問い合わせ

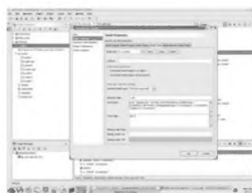
■サイバネットシステム（株）
TEL：03-5978-5453 FAX：03-5978-2201
E-mail：rinfo@cybernet.co.jp
URL：http://www.cybernet.co.jp/reflection/

●統合開発環境

Wind River Workbench 2.0

- オープンソースのEclipseフレーム・ワークをベースに設計され，VxWorksおよびLinuxをサポートしており，ほかのOSへの対応も可能．
- OSに依存することなく，開発プロセスを標準化することが可能．
- コード・ブラウジングと静的な分析をサポート．
- 複数のスレッド，プロセス，プロセッサの並行デバッグが可能．
- JTAG準拠のオンチップ・デバッグとターゲット・エージェントを使用したデバッグをサポート．
- Wind View技術に基づくシステム・レベル・プロファイリングが可能．

●価格：下記へ問い合わせ



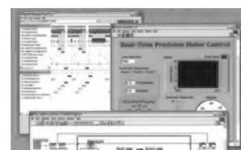
■ウィンドリバー（株）
TEL：03-5778-6001 FAX：03-5778-6003

●テスト開発用グラフィカル開発環境

LabVIEW 7.1

- モジュール式計測器に対応した，対話式アシスタント機能「Express VI」に加え，実行タイミング監視機能などのパフォーマンス最適化開発ツール，FPGA，PDAのほか，LabVIEW Real-Timeモジュールがデスクトップ PC上で実行できる機能などを搭載．
- 新たに5種類の「Express VI」を搭載したことで，迅速なテスト・アプリケーションの構築とデータ収録が可能．
- リアルタイム・アプリケーションで利用可能な「NI-DAQmx」は，シングル・ループのPIDアプリケーションの性能を30%向上させることができ，ハードウェア・タイミングでのループ実行を行えるなどの特徴をもつ．

●価格：¥158,000（ベース・パッケージ）
¥302,000（開発システム）
¥517,000（プロフェッショナル開発システム）



■日本ナショナルインスツルメンツ（株）
TEL：0120-527196 FAX：03-5472-2977
E-mail：prjapan@ni.com

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

7/14-16 **DIGITAL VIDEO EXPO EAST 2004**
Jacob K. Javits Convention Center, New York, NY, USA
CMP Media
http://www.dvexpo.com/east/

8/2-5 **LinuxWorld Conference & Expo**
Moscone Center, San Francisco, CA, USA
IDG World Expo
http://www.linuxworldexpo.com/

8/8-12 **SIGGRAPH 2004**
Los Angeles Convention Center, Los Angeles, CA, USA
SIGGRAPH
http://www.siggraph.org/s2004/

8/16-19 **COMDEX Korea 2004**
COEX Atlantic Hall & Conference Center, Seoul, Korea
COMDEX Korea
http://comdex.chosun.com/

国内イベント

6/28-7/2 NetWorld+Interop 2004 Tokyo
幕張メッセ(千葉県千葉市美浜区)
NetWorld+Interop 2004 Tokyo実行委員会
http://www.interop.jp/

6/30-7/2 第14回フラットパネルディスプレイ製造技術展
東京ビッグサイト(東京都江東区有明)
リードエグジビションジャパン(株)
http://www.ftj.jp/

7/6-8 国際光触媒テクノフェア2004
東京ビッグサイト(東京都江東区有明)
国際光触媒展実行委員会/(株)東京ビッグサイト
http://www.ptf.jp/

7/7-9 第7回組込みシステム開発技術展ESEC/第13回ソフトウエア開発環境展/第9回データウェアハウス&CRM EXPO/第6回データストレージEXPO/第1回情報セキュリティEXPO
東京ビッグサイト(東京都江東区有明)
リードエグジビションジャパン(株)
http://www.esec.jp/

7/12 STARC/ASPLA共同フォーラム
パシフィコ横浜(神奈川県横浜市西区)
(株)半導体理工学研究センター/(株)先端SoC基盤技術開発
http://www.semiconductorportal.com/st-as/

7/13-16 インターオプト'04
幕張メッセ(千葉県千葉市美浜区)
(財)光産業技術振興協会
http://www.oitda.or.jp/index-j.html

7/14-16 2004国際ウエルディングショー
インテックス大阪(大阪府大阪市住之江区)
(社)日本溶接協会/産報出版(株)
http://www.sanpo-pub.co.jp/CONTENTS/2004WELD/2004WELD.html

7/21-23 ワイヤレスジャパン2004/第1回次世代ワイヤレス技術展/第1回モバイル電子部品フォーラム
東京ビッグサイト(東京都江東区有明)
(株)リックテレコム/EJ.クラウス&アソシエート日本支社/YRP研究開発推進協会
http://www.ric.co.jp/expo/wj2004/pro_index.html

開催日，イベント名，開催地，問い合わせ先の順

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

レイヤの詳細解説からS-ATA IIの拡張機能まで，S-ATA技術の詳細解説と測定・評価法
開催日時　：7月1日(木)
開催場所　：オームビル(東京都千代田区神田錦町)
受講料　：58,485円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/sem/040701a.htm

C言語によるはじめてのLinuxプログラミング
開催日時　：7月1日(木)～2日(金)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：92,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/linux/clinux.asp

オブジェクト指向技術によるプロセス改善の実践
開催日時　：7月1日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

TCP/IPソケットプログラミング入門
開催日時　：7月2日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

初めてのVisual Basic 6.0
開催日時　：7月7日(水)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：29,400円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/VB/vb_beginner.asp

ロボット工学セミナー，人とロボットの交差点～ロボットデザイン～
開催日時　：7月9日(金)
開催場所　：工学院大学 新宿キャンパス11階第5会議室(東京都新宿区西新宿)
受講料　：12,000円(一般)，4,000円(学生)
問い合わせ先：(社)日本ロボット学会ロボット工学セミナー係，☎(03)3812-7594，FAX(03)3812-4628
http://www.rsj.or.jp/Seminar/2004/RSJ_Sympo_27.htm

はじめてのTCP/IP
開催日時　：7月17日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：12,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

PalmOSプログラミング基礎
開催日時　：7月22日(木)～23日(金)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：80,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/palm/palm.asp

情報セキュリティの基礎
開催日時　：7月23日(金)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

やり直しのための積分変換
開催日時　：7月24日(土)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

VC++&Win32APIによるマルチスレッドプログラミング
開催日時　：7月26日(月)
開催場所　：エイチアイ研修室(東京都目黒区東山)
受講料　：49,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)5773-8661
http://icp.hicorp.co.jp/seminar/c-vc/vc_multi.asp

ステート・マシンの設計技術
開催日時　：7月29日(木)
開催場所　：CQ出版セミナー・ルーム(東京都豊島区巣鴨)
受講料　：13,000円
問い合わせ先：CQ出版エレクトロニクス・セミナー事務局，☎(03)5395-2125，FAX(03)5395-1255

読者の広場

Interfaceへの声



2004年6月号特集「ようこそ二足歩行ロボット制御の世界へ」に関して

▷ 今回の特集は題名どおり，内容が入門編だったので，次は中級編，そして上級編でお願いします．ロボット技術者や学会の人間も読み応えのある内容を期待します．

（カーピー）

▷ 特集でうたっているように，二足歩行の制御方法をファームウェアの観点から具体的に細かく詳細に例を挙げて紹介してほしかった．どちらかというとハードウェア寄りであったのが残念！！（クロチャン）

アンケートの結果

興味のあった記事（2004年6月号で実施）

①第2章 二足歩行ロボットの制御回路の設計
②第1章 ロボット制御システムの構成と通信技術
③第5章 二足歩行ロボットの制御アルゴリズムとプログラミング
④プロローグ 二足歩行ロボットの現状
⑤第4章 ロボットの機構設計とサーボ・モータの選択
⑥Appendix 3 ロボットの機構とトルク設計
⑦第6章 ロボットに使われるセンサ技術
⑧第3章 CPLDを使用したRCサーボ信号発生回路の設計
⑨新発想のツール「Impulse C」を使ったソフトウェアのハードウェア化手法
⑩Appendix 1 フラッシュ・メモリの消去，書き込み，読み出し

特集『ようこそ二足歩行ロボット制御の世界へ』についてのアンケートの結果

Q1 あなたは今回登場したロボットのうちどれに興味がありますか？

ASIMO，QRIO，nuvo，HRP-2，トヨタ自動車の二足歩行ロボット，ながら

Q2 あなたにとってロボットのイメージはなんですか？アニメのヒーローや映画に登場したものでもかまいません．

鉄腕アトム，鉄人28号，ロボコップ，ターミネーター，ガンダム，パトレイバー，メカゴジラ

Q3 2005年4月から「愛・地球博」（通称：愛知博）が開催されます．あなたは会場に足をはこびますか？

①はい（38%） ②いいえ（0%）
③まだわからない（62%）

Q3 今後ロボットはどういうところで活躍すると思いますか？

①人間のペット（0%）
②掃除（6%）
③警備（25%）
④チャイルド・ケア（6%）
⑤介護（25%）
⑥接客（12%）
⑦その他（25%）
生体機能の代替/拡張，国防関係，産業用

Q5 あなたはロボットを作ってみたいですか？

①はい（88%） ②いいえ（0%）
③どちらともいえない（12%）

特集担当デスクから

☆T-KernelやTeacube，ミドルウェアの流通など，さまざまな部分で注目されているT-Engineだが，「組み込み開発キット」という特殊（？）な製品が一般ユーザにも購入できるという点にも注目したい．T-Engineを購入した人が「組み込み」のおもしろさに気付き，組み込み開発業界へと入って裾野を広げてくれることを期待しよう．

Interface

次号予告

2004年9月号は
7月24日発売です

今こそ基礎を身に付けチカラを上げる

原理から学ぶディジタル信号処理技術

DSPの変遷とディジタル信号処理技術への要求／FIRフィルタの設計と数値演算／ディジタル・フィルタを体験する～画像処理と音声処理／アプリケーション開発事例～通信処理／実装の心得

ディジタル信号処理技術は，通信や画像，音声，制御，計測など，幅広い分野で当たり前のように使われている技術である．そのため，今，この技術をしっかりと身に付けたエンジニアが求められている．

そこで次号特集では，ディジタル信号処理技術にフォーカスする．はじめにディジタル信号処理に特化したプロセッサであるDSPの変遷を通して，その時代ごとのディジタル信号処理への要求の移り変わりを振り返り，今，何が求められているのかを知る．次に原理となる数値計算やフーリエ変換，ディジタル・フィルタの作成といった基礎的な知識を徹底的に解説する．さらに，ディジタル信号処理を用いたアプリケーション開発の際のポイントについても紹介し，アプリケーションを実装する際の注意点や考慮すべき事がらについても述べていく．

編集後記

●経済産業省が5月にまとめた「新産業創造戦略」によると，先進的新産業群「燃料電池」，「情報家電」，「ロボット」，「コンテンツ」，ニーズ対応新産業群「健康・福祉・機器・サービス」，「環境・エネルギー・機器・サービス」，「ビジネス支援サービス」の戦略7分野に重点的に予算配分するようです．小誌の路線はまさにぴったり．（檀）

●そろそろ10年選手なのでドラム式洗濯乾燥機やらノンフロン冷蔵庫にするかなぁ～と思いつつ，まだ壊れてないのに買い換えるのももったいないなぁ～と悩む…かと思えば，テープの巻き戻しもろくにできなくなってきたデッキは，もう十分に棄てる覚悟はできてるのに，HDD/DVDレコーダ売り場でまた悩む（^^;）（M）

●↓がいきなり筋力を上げたいと言い出したので，会社に置いてある器具を貸し出すことになった．身体を鍛えるのは良いことだ．筋トレはやった分だけ効果が目に見えて現れるので，簡単にハマってしまって止められなくなる．欲が出てくるのである．さぁ，みんなで鍛えてギンギンのマッチョになろうぜ．（＝IO）

●↑の人の影響で，編集部内で筋力トレーニングがブーム．すっかり弱くなっていた上半身を，十数年ぶりに鍛えています．隣のT誌編集部をも巻き込んで，あちこちで筋トレグッズを使っている光景が展開しているわけですが，数か月後には，筋肉ムキムキになった編集部が見られるかもしれません!?（み）

●私鉄の自動改札機を通る際，パスネットカードとJRの定期券を間違えて入れてしまった．だって，外形寸法が似てるんだもん．それで，なぜか改札機を通れてしまった（窓口で清算したけど）．あと，JRの改札機でSuicaカードと会社のカード・キーを間違えるというボケをかましたけど，さすがに通れなかった．（もみ）

●伊豆にある河津七滝に行きました．その名のとおり大小七つの滝があり，それぞれの滝に行くのには数分間歩くだけ．私が気に入ったのは一番山奥にある釜滝．大きさ的には二番目ですが，滝のしぶきがかかるほど近くで見られるので大迫力．流行のマイナスイオンも浴びて体も心もリフレッシュできた…かな．（Y2）

●少し前に「年金」についての与党が出した改革案について議論があった．抜本改革ではないとする主たる理由は将来の出生率，経済成長率，年金基金の運用利回りなどの数字があまりにも楽観的であるということだが，膨大な「過去債務」を何とかするためには現在の受給者の受給額まで踏み込む必要があると思うし，そうすれば世代間の不公平感が少しは和らぐ気がする．（ちゃん）

●夏に向けダイエット開始．成功した友達に聞くとデンプン質のものを一切とらず，2週間で2kg痩せたそうだ．すごい！デンプン質というと，米・うどん・パスタ・いもetc…やっぱり私には無理だ．やはり面倒くさいけど運動しかないか….（かね）

お知らせ

■ 読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ 投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ 本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ 本書付属のCD-ROMについてのご注意

本書付属のCD-ROMに収録したプログラムやデータなどは著作権法により保護されています．したがって，特別の表記がない限り，本書付属のCD-ROMの貸与または改変，複写複製（コピー）はできません．また，本書付属のCD-ROMに収録したプログラムやデータなどを利用することにより発生した損害などに関して，CQ出版社および著作権者は責任を負いかねますのでご了承ください．

■ コピー・サービスのご案内

本誌バックナンバの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

●コピー料金（税込み）
1ページにつき100円

●発送手数料（判型に関わらず）
1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円

●送付金額の算出方法
総ページ数×100円＋発送手数料

●入金方法
現金書留か郵便小為替による郵送

●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数

●宛て先
〒170-8461 東京都豊島区巣鴨1-14-2
CQ出版株式会社 コピー・サービス係
（TEL：03-5395-4211，FAX：03-5395-1642）

■ お問い合わせ先のご案内

●在庫，バックナンバ，年間購読送付先変更に関して
販売部：03-5395-2141

●広告に関して
広告部：03-5395-2133

●雑誌本文に関して
編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版（株） 2004 振替 00100-7-10665
2004年8月号 第30巻 第8号 通巻第326号
2004年8月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏 村上真紀 山口光樹 大竹友美 小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森 祐子
広告／澤辺 彰 中元正夫 菅原利江

発行所／CQ出版株式会社 〒170-8461 東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122 FAX／（03）5395-2127
広告部（03）5395-2133 URL http://www.cqpub.co.jp/interface/
販売部（03）5395-2141 E-mail supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社 美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構） ISSN0387-9569